Automatic Extraction and Assessment of Entities from the Web

Dissertation

submitted in partial satisfaction of the requirements
for the degree of Doktoringenieur (Dr.-Ing.)

at
Technische Universität Dresden
Faculty of Computer Science

by
Dipl.-Medieninf. David Urbansky
born on November 25, 1984 in Dresden

Advisers:

Professor Dr. rer. nat. habil. Dr. h. c. Alexander Schill	TUD, Dresden, Germany
Associate Professor Dr. James Thom	RMIT, Melbourne, Australia
Professor Dr.-Ing. Michael Schroeder TU Dresden	TUD, Dresden, Germany

Day of Defense: October 15, 2012

Dresden, October 17, 2012

Statement of Authorship

This dissertation has been conducted and presented solely by myself. I have not made use of other peoples work (published or otherwise) or presented it here without acknowledging the source of all such work.

Dresden, July 9, 2012

David Urbansky

Abstract

The search for information about entities, such as people or movies, plays an increasingly important role on the Web. This information is still scattered across many Web pages, making it more time consuming for a user to find all relevant information about an entity. This thesis describes techniques to extract entities and information about these entities from the Web, such as facts, opinions, questions and answers, interactive multimedia objects, and events. The findings of this thesis are that it is possible to create a large knowledge base automatically using a manually-crafted ontology. The precision of the extracted information was found to be between 75–90 % (facts and entities respectively) after using assessment algorithms. The algorithms from this thesis can be used to create such a knowledge base, which can be used in various research fields, such as question answering, named entity recognition, and information retrieval.

Acknowledgement

This thesis was written between 2009 and 2012 during my time as a research assistant at the Chair of Computer Networks at the Dresden University of Technology. I want to thank my doctoral adviser Professor Alexander Schill for giving me the opportunity and the freedom to write my thesis about my own research interests. He also provided me with research positions in the projects Aletheia, X2Lift, and Intellix which financed me during the three years of writing my thesis. I am very grateful for the critical feedback that I received from Associate Professor James Thom. Although we only met twice during these three years, our day-long meetings and feedback over email were of great help to me and shaped my thesis substantially. Also, I want to thank my adviser Dr. Daniel Schuster for constant feedback on my progress, valuable tips from his experience, and insightful suggestions.

During my time at the Faculty of Computer Science, I supervised over thirty students. That alone was an incredible experience from which I learned a lot about providing help, working with different characters, and also myself. Some research areas in this thesis were studied in detail in diploma theses of my students. I want to thank Christopher Friedrich, Christian Hensel, Marcel Gerlach, Dmitry Myagkikh, Martin Werner, Robert Willner, and Martin Wunderwald for writing their master theses under my supervision and producing outstanding results.

I want to express my gratitude to my colleagues whom I was lucky enough to be working with. My thanks go to Marius Feldmann who led me into the PhD program in the first place. Furthermore, I highly appreciate working with Philipp Katz, Klemens Muthmann, and Sandro Reichert on the Java toolkit “Palladian”, our joint effort in giving something practical back to the information retrieval and machine learning community.

And of course I want to thank my parents for always being there for me even though they have no idea what this thesis is all about. Lastly, I want to express my appreciation to my lovely girlfriend Crystal, who not only read the thesis back to back several times, correcting all my grammar and spelling errors, but who was also patient and understanding with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web is a huge repository of distributed information. Users of the Web create millions of Web pages on a daily basis and the added information is highly diverse. Users can find scientific papers, product reviews, news, videos, travel guides, and various other types of information. The Web has evolved into a dynamic corpus in which users cannot only consume information, but can also actively participate by booking hotels, playing browser games, commenting on blogs and forums, drawing images, and much more. Currently, the Web is primarily structured for easy user access, that is, Web pages are structured and displayed for human users who consume or interact with the information. To provide the users with even more sophisticated services that require information from multiple sources, machines need to be able to read, "understand", and interact with distributed data. For this purpose, Web Services using the Simple Object Access Protocol (SOAP) (Mitra and Lafon, 2007) and Representational State Transfer (REST) (Fielding, 2000) were developed to allow data exchange. These services are often used to create so-called mashups. "Mashups" are applications that use data and services from different sources to create new value for a user. These mashups, however, are often hard coded by a human programmer because the data from the services must be understood by a human and aggregated in a way that it can be integrated. This approach is, however, highly unscalable but should be solved with the Semantic Web. The vision of the Semantic Web (Berners-Lee et al., 2001) is to make the Web more machine readable and to allow machines to integrate data from different sources automatically following predetermined schemata and ontologies. One major goal over the next few years will be to enhance, annotate, and structure available unstructured information to make it more easily accessible for machines. This development would allow for much more sophisticated applications, such as natural language question answering.

In the following sections, we will describe the motivation, requirements, focus and limitations of this work.

1.1 Motivation

Recent studies have shown that between 40 % (Pound et al., 2010) and 71 % (Guo et al., 2009) of user queries on search engines contain entities such as movies, people, and organizations.

We can call those queries entity-centric. “Entity-centric” denotes all information related to entities. It is therefore crucial that the search engine recognizes these entities in order to provide the sought-after information. In the following sections, we outline scenarios that require a knowledge base of entities and information related to those entities.

1.1.1 Question Answering

Information on the Web is highly distributed and most often not made for consumption by machines. Much information is redundant, while other types of information can only be found once (Afanasyev et al., 2011). From the Web user’s point of view, this information scattering is not always beneficial. Imagine a user wants to inform himself about a specific movie. He would need to search for it, click through the returned Web pages, and aggregate the listed information manually. Some Web pages may have information about the cast of the movie, but do not have any images. Another Web page may list related movies, whereas yet another page may have products and reviews associated with the movie. The user must go through a time-consuming process of searching and aggregating the information he finds on the Web pages. Having a single access point for the user to look for information about the movie of interest would significantly save the user’s time. Such a single access point could be a Web page that enables the user to search for a general **concept** (a type of entity, for example *Movie*) or a specific **entity** (an instance of an entity type, for example, *The Dark Knight* as an instance of the concept *Movie*). The knowledge base that the user queries must be up-to-date, that is, if the user’s query is valid and information about the queried entity exists on the Web, the knowledge base must return this information. We will consider questions about the movie *The Dark Knight* as an example scenario to better explain the possible needs that a user will have. Users might also have certain questions about entities and do not want to see all available information. For example, a user might ask “Who is the director of The Dark Knight?”, in which case a system that understands the semantics of the question and knows about the entity would be able to answer the question directly with *Christopher Nolan* instead of providing a set of documents with possible answers.

Knowledge Base vs. Search Engine vs. Database

It is important to elaborate on the difference between a knowledge base, database, and search engine. We define a knowledge base as information storage that can be queried and returns machine-readable information, or as May (2001) explains it, “A knowledge base is not a static collection of information, but a dynamic resource that may itself have the capacity to learn, as part of an artificial intelligence expert system”. A database is an organized collection of data without the learning capabilities of a knowledge base. The main difference between a knowledge base and a search engine is that when a user queries a search engine, he gets a ranked list of documents that include the search terms of his query, while a knowledge base would try to answer the user’s information need directly. Consider a user searching for *Mel Gibson* using a search engine and a knowledge base front end. The search engine would return **documents** (usually Web pages) that are about *Mel Gibson*, while a knowledge base is expected to compute or retrieve an **answer** in form of facts about *Mel Gibson*, a list of his movies, and so on. In this thesis, we create a knowledge base, not a search engine.

Types of Information

The information that might interest the user depends on the concept of the entity. The following information types are typical in Web information retrieval:

- **Facts and Relations**: Almost every entity has attributes and thus facts attached to it. A fact is a statement that holds true in the context of the entity. A relation is also a fact, but it points to another entity instead of a literal. Facts provide a quick and easy summary of an entity. Instead of reading long paragraphs in which the statements are dispersed across the text, users often prefer to see those facts directly. In our example scenario about the movie The Dark Knight, the facts that might interest the user are *runtime*, *aspect ratio*, and *release date.*
- **Images**: Users often search the Web for images. We can classify images along three dimensions: their semantic content (image about people, trees, or festivals), their serialization (JPG, SVG, or PNG), and their source type (photo, drawing, computer-generated image). If the user searches for information about a person, he might be interested in photos of this person. However, he may also want to find paintings or vector graphics of the person. In our example scenario, the user might want to see photos from the premiere party, images of the stars, and behind-the-scenes shots related to the movie *The Dark Knight.*
- **Videos**: Videos are popular objects for information retrieval. We can classify videos along the same dimensions as the images. In our example scenario, the user might be interested in seeing a trailer, a "making of" video, or an interview with the director of the movie.
- **Audio**: Users might also be interested in retrieving audio content. Over the last few years, podcasts have spread and become popular, but music, audio lectures, or simply sounds might also fulfill the information needs of the user. In our example scenario, the user might want to hear the soundtrack of the movie or a critic's review of the movie.
- **Interactive Content**: Interactive content refers to applications that enable users to engage with the content. Interactive content is most often delivered in proprietary formats, such as Adobe Flash movies, Java Applets, or Silverlight Applications. Traditional information retrieval has mainly focused on image, audio, and video, but the widespread availability of interactive content demands a retrieval process for this type of information as well. In our running example, the user might want to play a Flash game about the movie or take part in a quiz delivered in Silverlight.
- **Products**: Today, over 72 % of Web users shop online (Reese, 2011). Clearly, products are of major interest to users. Almost everything that can be purchased offline can be bought online as well. In our scenario, it is easy to imagine that a user is interested in buying the Blu-ray of the movie, a movie poster, the book upon which the movie is based, merchandise, or the soundtrack on CD.
- **Services**: People to not only buy products but also services online. In our scenario a service could be a streaming service to download the movie or a ticketing system which lets users buy tickets to watch the movie in a cinema.

- **News**: Many people like to stay informed and read the news frequently. Since there are so many news items every day, publish-subscribe mechanisms such as Really Simple Syndication (RSS) and Atom have become popular means of receiving information from selected sources. There is still a lack of information services that enable users to specify their information needs; instead, users must subscribe to all news stories from a certain source. In our example scenario, it would be beneficial for the user if he could subscribe to the single entity *The Dark Knight* and receive only relevant news (for example, a DVD release, remake filmed, sequel planned, et cetera), instead of subscribing to the complete movie channel.
- **Opinions and Reviews**: Often users are interested in what others say or think about the entity. This does not apply to all searches, since a user searching for "Penguin" most likely does not want a stranger's opinion about penguins. Many searches, however, pertain to man-made objects such as music albums, cars, movies, and so on. For most of these man-made objects, there are reviews and opinions. In our example scenario, the user searches for an entity that is also a product. He might therefore be interested in how many people would recommend the movie or might want to read a review about it.

Figure 1.1 depicts a mockup of a system that combines different types of information. In this thesis, we will describe extraction techniques for each of the information types in the figure. Chapter 10 shows our prototypical implementation of this mockup.

The question answering use case is the main use case that we will pick up again in Chapter 9, when we show how we apply question answering techniques for structured and semi-structured sources to answer user questions.

1.1.2 Entity Dictionary

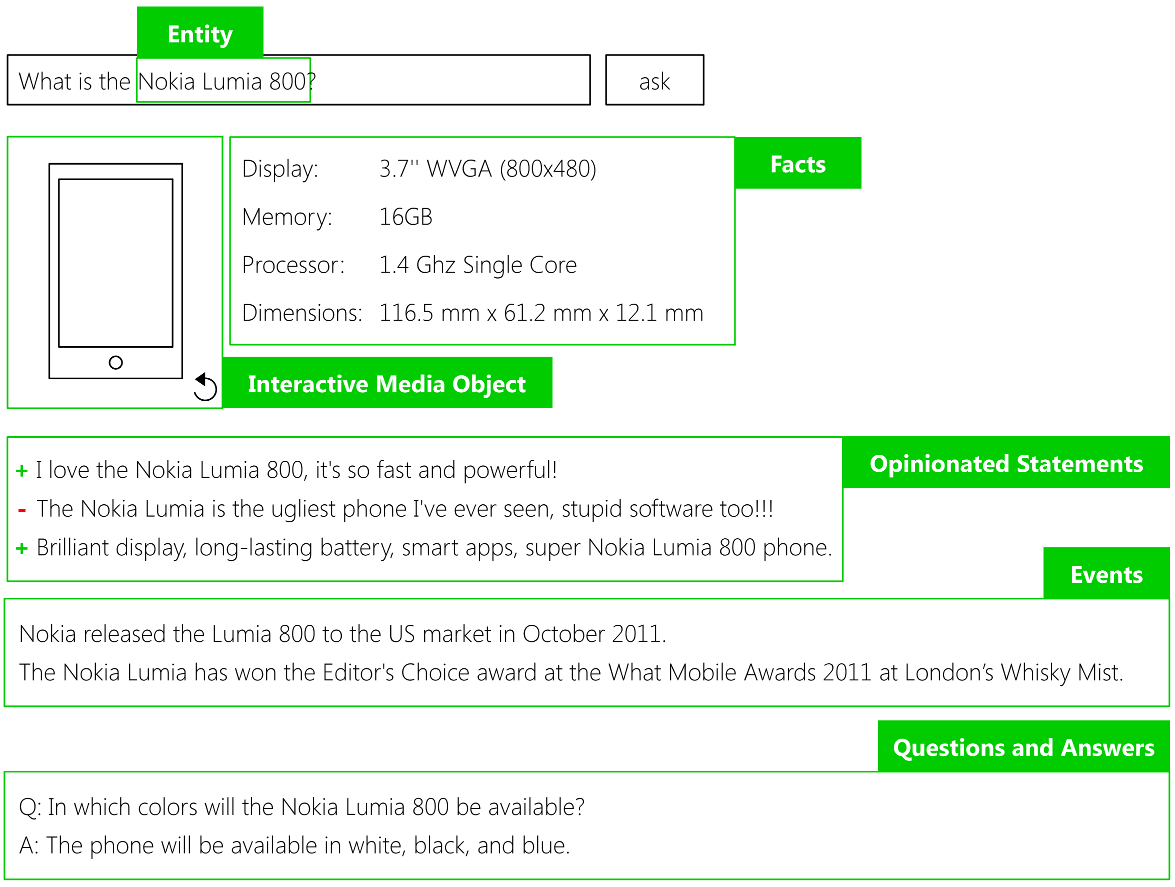
Dictionaries about entities and their relations to each other are beneficial for a large variety of applications; in particular, they help improve results in named entity recognition (NER) (Chinchor, 1997). Named entity recognizers recently became available through Application Programming Interfaces (APIs) such as OpenCalais[1] or AlchemyAPI[2], underlining their importance for the industry.

The following two use cases outline how such an entity dictionary can be used within the context of named entity recognition.

Semantic Search

The ability to recognize entities that belong to concepts allows one to semantically tag text. A search agent can then find results for the search query not only on a keyword basis, but also semantically. This function is especially helpful when an entity name is highly ambiguous, as many person names are. The first type of ambiguity is an "inter-concept" ambiguity. For

[1]`http://www.opencalais.com/`, last accessed on 6th of May 2012
[2]`http://www.alchemyapi.com/`, last accessed on 6th of May 2012



Display:	3.7" WVGA (800x480)
Memory:	16GB
Processor:	1.4 Ghz Single Core
Dimensions:	116.5 mm x 61.2 mm x 12.1 mm

+ I love the Nokia Lumia 800, it's so fast and powerful!
- The Nokia Lumia is the ugliest phone I've ever seen, stupid software too!!!
+ Brilliant display, long-lasting battery, smart apps, super Nokia Lumia 800 phone.

Nokia released the Lumia 800 to the US market in October 2011.
The Nokia Lumia has won the Editor's Choice award at the What Mobile Awards 2011 at London's Whisky Mist.

Q: In which colors will the Nokia Lumia 800 be available?
A: The phone will be available in white, black, and blue.

Figure 1.1: Mockup of a Entity Information System Combining Different Pieces of Information

example, a user could search for *Sam Smith* and the search engine could ask him whether he wants to find more information about Sam Smith the rugby player, the football player, or the sportswriter. After the user chooses an option, the results will no longer be mixed, allowing the user to find what he needs faster. The second type of ambiguity is an "alias ambiguity", where different aliases all refer to the same entity. For example, a user searching for *James Eugene Carrey* can now expect to get results where *Jim Carrey* is mentioned, since they refer to the same entity.

Information Enrichment

Imagine the following scenario: A user browsing the Web runs across the name of a movie he has never seen. He might leave the page and search for the movie using a search engine to learn more about it. If we have a large knowledge base with semantic information about entities (such as movies or people), we can recognize and enrich these entities for a human user. In the described scenario, we could find additional information, such as the *director* or the *release date* of the mentioned movie, and provide the user with this information directly on the Web page mentioning the entity. Since the information from the knowledge base is semantic, we could also disambiguate entity names, that is, we would provide information about the person *Queen Elizabeth II* on a Web page about the English queen and information about the ship *Queen Elizabeth II* on a Web page about travel and cruise ships. This information enrichment could happen using browser extensions that analyze the Web pages and enrich detected entities on the client.

1.1.3 Dataset for the Semantic Web

In the vision of the Semantic Web, the information on the Web becomes more machine readable. The realization of this vision would allow for many new applications that would benefit the human end user. Today, many domain ontologies have been created and connected (DBpedia, MusicBrainz, et cetera) in order to allow machines to reason over that information and gain new insights for the human. The desired knowledge base will connected to existing ontologies, and thus, be a part of open information for the Semantic Web. Figure 1.2 shows a large part of the Linked Data Cloud available on the Web. Each circle represents a dataset. The diameters of the circle hint at the size of the dataset (the relation between circle diameter and the dataset size is not exactly correct). An arrow from one dataset to another means that the source dataset links to the other dataset. There are three kinds of datasets: (1) one that has only outgoing references to other datasets, for example the Gov Track dataset, (2) a dataset that has only incoming references, such as Sem Web Central, and (3) datasets that have in- and outgoing references, such as DBpedia. Datasets with many incoming references, such as Music Brainz, DBpedia, ACM, and Gene ID, are very popular and thus their vocabulary should be used by new datasets within the same domain. As of end of 2011, there are over 31 billion Resource Description Format (RDF) triples with more than 504 million links between them (Franzon, 2011). These numbers are growing exponentially[3].

1.2 Requirements

This thesis focuses on the (semi-) automatic construction and maintenance of a domain-independent knowledge base with the scenario of question answering in mind (as described in Section 1.1.1). From this scenario, we derive the following requirements for the system which we will call WebKnox (**Web Kno**wledge E**x**traction) from now on.

Domain Independence

The knowledge base must be domain-independent. "Domain" describes a certain high level area of knowledge. For instance, "products", "nature", and "travel" are all domains.

Domain-independence means that all techniques and approaches chosen to construct and update the knowledge base do not rely on the aspects of a single domain.

[3]The number of datasets has almost doubled annually starting from 2007, see `http://richard.cyganiak.de/2007/10/lod/`, last accessed on 25th of March 2012

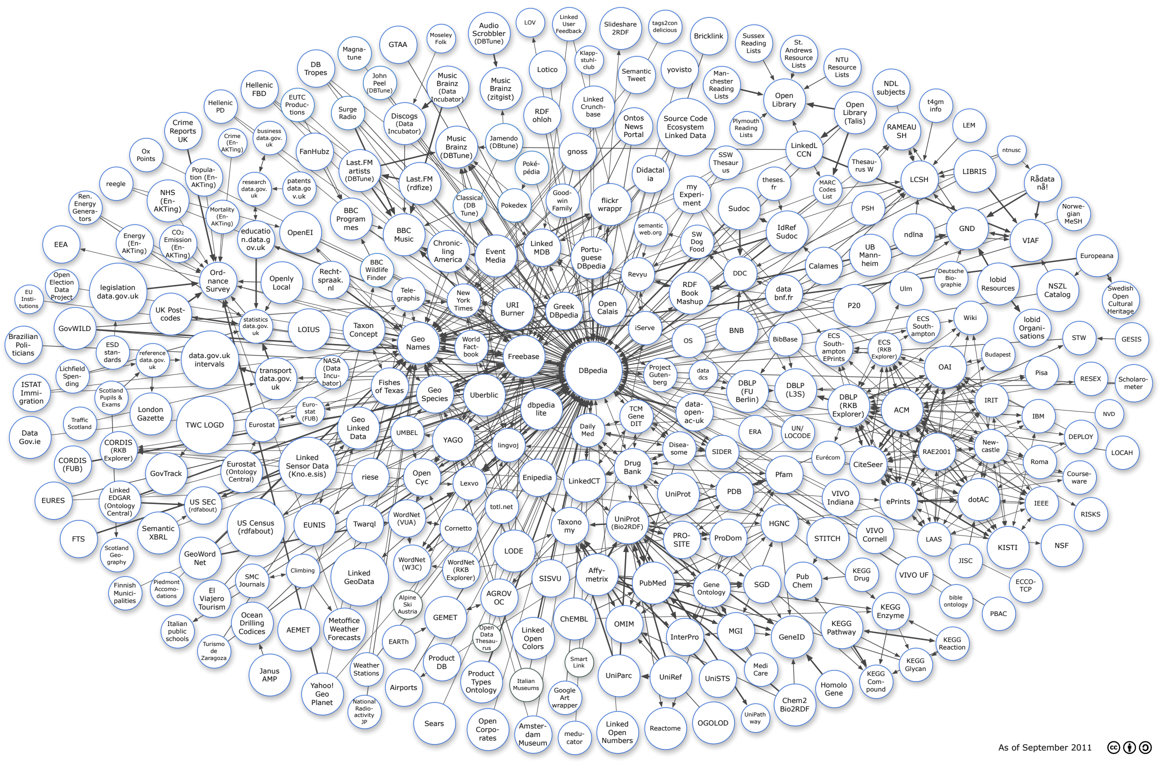


Figure 1.2: Linked Data on the Web as of September 2011 (Cyganiak and Jentzsch, 2011)

Accuracy

The knowledge base must be accurate, that is, the reliability of the information must be very high. Since automatic extraction of information will bring a certain amount of uncertainty into the knowledge base, we must assign each information piece a confidence score, which we call “trust”. Depending on the application, “very high” accuracy means that in 80–100 % of the cases, information must be completely correct and reliable.

Up-to-Date

We want to continuously expand the knowledge base. While some concepts consist of a rather fixed set of entities, others are very dynamic and new entities appear daily. For example, it is rarely necessary to search for new entities belonging to the concept *Country*, since the number of countries in the world rarely changes. On the other hand, the concepts *Movie* and *Mobile Phone* are highly dynamic and new entities are mentioned very often. Current entity extraction techniques focus on extraction of entities belonging to static concepts. For example, there are techniques that search for lists of entities for a certain concept and extract these entities without considering searching again. We need to automatically discover entity-rich resources and check them periodically for new entity mentions.

Semantic

Information in the knowledge base must be semantic. To define the term “semantic” in this context, it helps to understand the differences among data, information, and knowledge. Since there are no agreed-upon definitions of these three terms, we explain these terms again in the context of this thesis. Figure 1.3 shows the three terms in relation to each other.

Data is pure syntax without semantics, that is, there is no meaning to the data. Consider this data: `21, 25, 24, 32, 34, 31, 35`. Without any interpretation and meaning, these numbers could refer to anything. We could think of it as temperature data for one week or the ages of a group of people.

As soon as we can construct **relations** between data and have an **interpretation** of the data we can add meaning to it and have information.

Information is understood data, or data with meaning. For example, if we know that the data series above is one week of temperature data, we have information.

When we **aggregate** information, put it in **context**, and are able to **reason** over it, we have knowledge.

Knowledge is usable information in a certain context. For example, if we have information about temperatures during a year and information about the growth speed of trees in the context of researching tree populations, we can use and relate the available information to gain new insights (reasoning) such as that the temperature has an influence on tree growth speed.

Semantic refers to the meaning of data. We can extract data from the Web without “knowing” what it means. Such a database of terms would not be semantic. Semantic in our knowledge base context means:

- **Disambiguation of senses**: A term (word or number) can have several meanings. “Queen Elizabeth” might refer to the English Queen, a person, or to the cruiseship, depending on the context in which the term appears. Similarly, 21 might mean 21 Euros (amount of money) or 21 °C (temperature) also depending on the context.

- **Disambiguation of entities**: Even when the sense of a term is understood, it can still refer to several entities. For example, names of people are highly ambiguous. “Sam Smith” is a football player, rugby player, or sportswriter. The correct entity must also be disambiguated using the context.

- **Relations**: A term often has more information related to it. Words can have synonyms and homonyms; entities can have facts, alternative names, and relations to other entities. For example, the string “Jim Carrey” refers to the actor *James Eugene Carrey*. Having facts about *Jim Carrey*’s age, his list of movies, and his biography in the knowledge base would enrich the semantics behind the string “Jim Carrey”.

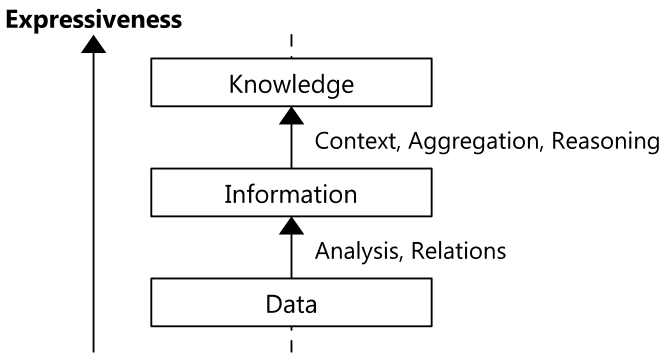


Figure 1.3: Differences among Data, Information, and Knowledge

1.3 Focus and Limitations

We have outlined several scenarios in which the knowledge base that we want to obtain automatically would be beneficial. Since we cannot solve all the problems associated with these use cases, we need to sharpen the focus and limit the scope.

Focus

The goal is to extract many information types mentioned under Section 1.1.1. The main focus lies, however, on the **entity extraction** and **assessment**, as it is the basis for all other extraction steps. We will therefore search for techniques to extract entity names with high precision from the Web. The second most important focus of this thesis is **facts** because we can build a Linked Data dataset using entities and factual information. We focus on the extraction of facts with only one correct value (for example, facts for the attribute *filming location* of a movie can have several correct values). Although there is a plethora of information on the Deep Web (Web pages hidden behind forms of databases, usually not indexed by search engines), in this thesis we focus on the **Visible Web** (easily accessible Web pages made to be consumed by humans) and the **Semantic Web** (Web resources made to be consumed by machines) for extracting information.

Limitations

The first limitation is that we can only extract entities that are mentioned on the Web. The popularity of the entities or their concept also play an important role in extraction quality. For example, on the Web, names of movies are more popular than names of ornithologists, which will likely increase the precision with which we can detect movie names. We do not extract multiple values for attributes although that might be possible in reality, for example, the *cast list* attribute of a movie can have multiple correct values – actor names in this case. We also do not try to build the biggest knowledge base, but rather show that a knowledge base can be built automatically from multiple sources with high accuracy. The algorithms used in this thesis are limited to the **English** language. Extractions from other languages are not considered. For most parts, we also do not address efficiency issues and speed of algorithms, but only effectiveness. Exceptions to this limitation are the feed update strategies and the focused crawler for question and answer extraction.

1.4 Research Questions and Hypotheses

This section covers the research questions we try to answer throughout this thesis. For each question, we state a hypothesis that we will evaluate in later chapters.

Which techniques can be used to extract entity mentions on the Web and how well do they perform?

Entities can be found in different structures on the Web. We need to find out which extraction techniques are the best to extract a large amount of entities in a precise manner.

Hypothesis: Entities can be extracted from plain text using patterns and named entity recognition techniques. Furthermore, we can employ wrapper and bootstrapping techniques to increase the number of extracted entities. These techniques include algorithms for extracting entities from lists and tables. Using a variety of different techniques on arbitrary Web pages will yield entity extractions that are not available in DBpedia.

How can we efficiently poll Web sources to extract new entities?

New entities (for example, products) appear on the Web with a high frequency. To keep a knowledge base up-to-date, we need to find efficient ways to poll Web sources in order to apply extraction techniques.

Hypothesis: We can automatically read news feeds and apply a prediction strategy to efficiently read news only when new messages are posted but without leaving out too many opportunities to extract entities. We claim that using a moving average algorithm on the update intervals of feeds will outperform fix polling intervals and algorithms relying on the post distribution of feeds. We measure the quality of the update strategies regarding the delay and the bandwidth consumption.

How can we ensure high precision of the extracted information?

Extracted information is by its nature noisy. To increase the quality of the final knowledge base, we need to find mechanisms to effectively assess extracted entities and filter out incorrect extractions.

Hypothesis: Given a small set of training data for each concept, we can employ a set of machine learning algorithms to classify extracted information. Using a mixed set of features, such as the extraction redundancy, the types of extraction sources, and generic language features, we can yield a higher classification precision than reported in related work on the same problem.

What entity-centric information is useful for question answering and how can we extract it?

One use case of the knowledge base should be question answering. In order to answer user questions we need to find out which types of information interest users and how we can extract this information and store it in the knowledge base.

Hypothesis: Users benefit from additional multimedia with their answers. We can use a keyword-based approach to find entity-related interactive multimedia objects and images. Using a machine learning-based news extractor will furthermore link entities to related news items. We can use a text classification approach to detect positive and negative sentiments in about 90 % of the cases for short statements.

How do ontology-based question answering systems compare to Web extraction-based question answering systems?

The knowledge base can be used as an RDF repository for facts. This repository can be used for a question answering scenario. Thus, we need to find out what percentage of questions can be answered using an ontology-based question answering system compared to systems that extract answers from the Web without requiring an ontology.

Hypothesis: Fewer questions can be answered using an ontology-based question answering system. The answers will, however, be more accurate.

1.5 Summary

We have now motivated use cases that would benefit from a knowledge base of entities and entity-related information. Furthermore, we explained which requirements such a knowledge base must meet and posed our research questions and hypotheses, which will be answered in the following chapters.

The problem we are trying to solve in this thesis is the **automatic** generation of an **accurate knowledge base of entities and entity-centric information**. As we will see in the next chapter, existing approaches for building such a knowledge base rely either on user input, extract information from a single or only few websites, or are very inaccurate.

The remainder of this thesis is structured as follows. First, we start by providing definitions, background information about knowledge bases, and explanations of evaluation measures in Chapter 2. In Chapter 3, we then introduce an architecture for our Web information extraction system by describing its main components, which will be explained in more detail throughout the thesis. Since our goal is to retrieve information in a fast manner, we begin by describing a retrieval strategy for Web feeds in Chapter 4. In Chapter 5, we review related work on entity extraction from the Web, describe five extraction techniques that are used by our system in detail, and compare them against each other. We will see that the extraction results are still very imprecise, which leads us to Chapter 6, which explains different approaches for assessing uncertain extractions. The goal in this chapter is to find an algorithm that filters incorrect extractions to improve the precision without sacrificing the recall. Chapter 7 then details five techniques to extract factual information about entities from the Web. These techniques are essential in building the large knowledge base that we envision. In Chapter 8, we describe further extraction techniques for additional, entity-centric information, such as news, opinionated statements, and interactive multimedia objects. These extraction objects are valuable for an entity-centric knowledge base as we have motivated earlier. Chapter 9 reviews, develops, and compares question answering approaches. Our goal in this chapter is to find out whether knowledge bases are better suited for natural language question answering than systems that try to find answers on the Web. In Chapter 10, we showcase some examples of practical applications that can be developed using the information from the knowledge base that we build in this thesis before we conclude with the results and findings in Chapter 11.

Chapter 2

Background

This chapter explains the basic terminology and ideas necessary to understand the next chapters in this document. First, we will briefly describe information retrieval and define the most important terms used throughout this thesis. Second, we will distinguish between three kinds of sources on the Web since we make a differentiation in our algorithms later on. In the last section of this chapter, we give an overview of the related knowledge bases and information extraction systems. A more detailed review of several of these systems follows later in the pertinent sections.

Most algorithms and approaches explained and used in this thesis belong to the field of information retrieval. Information retrieval is a field of research that is concerned with searching and ranking documents matching a user query. Sources used for answering the query can be structured (such as databases or RDF repositories) or semi-structured (such as Web pages). Information retrieval employs approaches from many disciplines such as statistics, linguistics, information architecture, and information science. We will especially make use of the statistic and linguistic aspects in this thesis.

2.1 Definitions

In this section, we define the terms concept, named entity, attribute, and statement, as they are fundamental to other topics covered in this chapter.

2.1.1 Concept

A concept is a chunk of text that refers to "an abstract or general idea inferred or derived from specific instances" (Stark and Riesenfeld, 1998)[1]. A concept is therefore a class of things and it can be instantiated with a specific instance – an entity. We use the term "concept" throughout this thesis. Other researchers also use the term "entity type" which is a synonym to the term "concept" in the scope of this thesis. We will use both terms interchangeably.

[1] `http://wordnetweb.princeton.edu/perl/webwn?s=concept`, last accessed on 20th of June 2012

2.1.2 Named Entity

There is no consensus on the definition of a named entity in the research community. Often, only instances of concepts in a certain scenario are considered named entities. The following definition is taken from the named entity recognition task from CoNLL 2002:

"Named entities are phrases that contain the names of persons, organizations, locations, times, and quantities." (Sang and Meulder, 2003a).

The CoNLL 2002 definition is useful when clarifying the goals of the NER task. It also unnecessarily limits named entities to the concepts *Person*, *Organization*, *Location*, *Time*, and *Quantity*. Also, it is arguable whether time and quantity are real named entities. The term "named" restricts the task to entities that are rigid designators as defined by Kripke (1981) who says that "A rigid designator designates the same object in all possible worlds in which that object exists and never designates anything else" (LaPorte, 2006). Following this path, however, leads us into a philosophical discussion about what an entity is. Nadeau (2008) emphasizes how difficult the definition of a named entity actually is. He says, his definition is "[...] ugly and circular, but [...] practical!": "The types recognized by NER are any sets of words that intersect with an NER type" (Nadeau, 2008).

The Message Understanding Conference (MUC) definition states that named entities are: "proper names, acronyms, and perhaps miscellaneous other unique identifiers" which belong to one of the following types "*Organization*: named corporate, governmental, or other organizational entity *Person*: named person or family, and *Location*: name of politically or geographically defined location (cities, provinces, countries, international regions, bodies of water, mountains, et cetera)" (Chinchor, 1997). Again, we see a very vague definition of what named entities are.

Borrega et al. (2007) only considers nouns or noun phrases whose referent is unique and unambiguous to be named entities. Furthermore, they distinguish between Strong Named Entities (SNE) and Weak Named Entities (WNE). SNEs are "formed by a word, a number, a date, or, in some cases, a string of words referring to a single individual entity in the real world". WNEs are syntactic elements consisting of at least one proper noun. Borrega et al. (2007) provide a collection of guidelines to determine what entities fall into the groups SNE and WNE. They base their guidelines on Spanish tests and admit that their definitions and guidelines always have exceptions.

The organizers of the TREC 2010 entity track also call defining entities on the Web an unsolved problem (Balog et al., 2010b), and Rössler (2007) reiterates that there is no consensus on the definition of named entities. He concludes his research on the definition of named entities stating that there are often no definitions but only guidelines. MUC and Automated Content Extraction (ACE), for instance, have refactored their guidelines for named entity tagging due to the difficulty of this task. The definitions, or rather, the guidelines, should be tailored to the context of the application.

We will therefore define the term "entity" for the scope of this thesis. Some researcher also call an entity that can be extracted from the Web a "Web object" (Nie et al., 2007). We define a named entity as follows: "An entity is a collection of names that refer to exactly one or to multiple identical, real or abstract concept instances. These instances can have several aliases and one name can refer to different instances. A 'named entity' is a reference to an entity using one of the entity's aliases."

Figure 2.1 shows the relation between concepts and entities. The figure allows us to explain our definition using some examples.

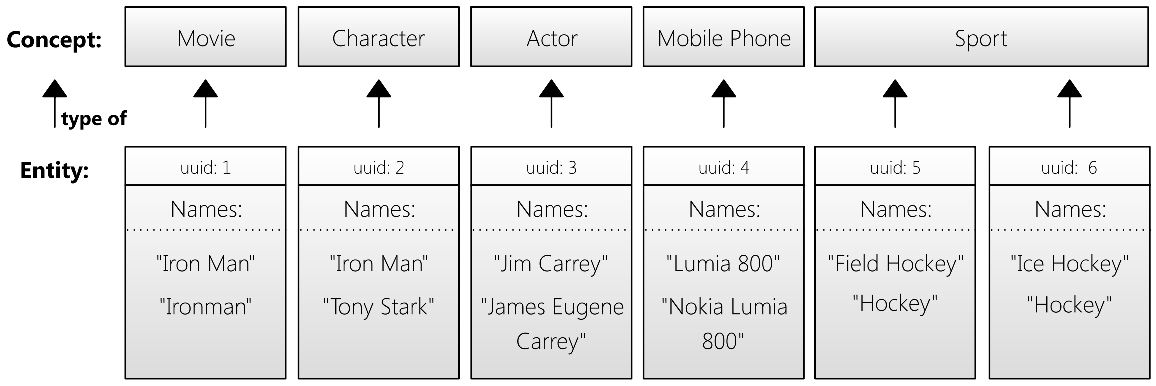


Figure 2.1: Relationship between Concepts and Entities

Ambiguity We can see that the movie entity *Iron Man* has another alias named *Ironman*, which actually refers to the same entity, that is, their uuids are identical. Furthermore, we can observe that the name "Iron Man" is ambiguous and might also refer to the Marvel comic character. Names are often ambiguous and need to be disambiguated in the context in which they are mentioned. The names must be rigidly designated; the name "the 2008 movie where Robert Downey Jr. plays a comic hero" is therefore not an entity. Due to their ambiguity, all entities must get a universally unique identifier (UUID) (Leach et al., 2005).

Table 2.1 shows four example entities classified along two dimensions.

	Abstract	**Concrete**
Specific	$1,000,000	Jim Carrey
Generic	Field Hockey	Lumia 800

Table 2.1: Example Classification of Entities According to our Definition

Generic and Specific *Jim Carrey*, the actor, refers to exactly one real world instance, while the mobile phone *Lumia 800* refers to multiple similar real world instances. People are specific in the sense that a concrete entity exists only once in the real world. Products are generic. For example, the mobile phone *Lumia 800* exists multiple times in the real world so they are "identical concept instances". We are not interested in these different instances, but rather in their common name, since they all share the same attributes, such as *display size*. Many concepts are generic, such as gene, car, or movie names. More information about specific and generic entities can be found in the work by LingPipe (2007).

Abstract and Concrete While *Lumia 800* refers to a collection of real world objects, the sport entity *Field Hockey* is an abstract instance. Playing hockey makes it real, but until then it is an abstract instance of a concept. It is not a concept according to our definition since there are no instances of *Hockey* itself, but only instances of hockey games. The same is true for instances of event concepts, such as *Concert* or *Conference*. Our definitions also allow us to have temporal and numerical instances, such as *$1,000,000* as abstract entities.

2.1.3 Attribute

An attribute is a modifier for a concept. A concept can have multiple attributes and one attribute can belong to multiple concepts.

Attributes have a domain and a range. The domain specifies the concepts they modify and the range specifies the range of values. For example, the attribute *display size* belongs to the concepts *Mobile Phone* and *Notebook* (domain) and can have values between 1 and 25 inches (range). Multivalued attributes are beyond the scope of this thesis.

2.1.4 Statement

Statements are assertions about entities. While attributes modify concepts, statements are assigned to entities. We can classify statements along the two dimensions “truth” and “representation”. Along the “truth” dimension, such assertions can be falsehoods, facts, or opinions. Along the “representation” dimension, statements can be unstructured (natural language) or structured (for example, RDF). Table 2.2 provides examples of statements.

	Falsehood	**Opinion**	**Fact**
Structured	`<earth>` `<hasForm>` `<flat>`	`<earth>` `<is>` `<gorgeous>`	`<earth>` `<hasForm>` `<ellipsoid>`
Unstructured	The earth is flat.	The earth is gorgeous.	The earth is an ellipsoid.

Table 2.2: Classification of Statements

2.2 Sources of Information on the Web

Our goal is to extract information, primarily in the form of named entities and facts from the Web. Before we can develop techniques and algorithms to extract the desired information, we need to analyze which types of sources we can find on the Web, how they are structured, and how frequently they appear. In the following sections, we only analyze textual documents on the Web; an information source can, however, be a Web user who interacts with a Web application. For example, if a user searches for *Jim Carrey* using the search engine Google, the user becomes an information source and intelligent algorithms could learn the actor entity *Jim Carrey* from his input.

2.2.1 Types of Structures

Textual documents on the Web can be classified into three major categories: unstructured, semi-structured, and structured documents. Across different research domains, definitions of these types vary (Chang et al., 2006) and often there are no exact distinctions. We will therefore explain how we understand the different types in the context of this thesis.

Unstructured Information Sources

Unstructured sources have little or no structure to help a machine parse the data. A simple text paragraph or this very sentence is called “plain text”. Both contains little or no markup for machines and are thus considered unstructured. Note that even text is not completely unstructured; sentences are made of words which are again made of letters. Words are most often separated by white spaces and a sentence usually ends with a period. As soon as the author of a text uses symbols to encode more advanced structured parts such as lists or tables (as often found in wiki-style texts), the text is no longer unstructured.

Semi-structured Information Sources

Semi-structured sources must have some markup that can guide a machine that is extracting information. In Web information extraction, semi-structured sources are mainly HTML files. These files often contain a great deal of unstructured data in texts, but use tags to structure this data for rendering purposes (Chang et al., 2006). The HTML structure can be parsed as a Document Object Model (DOM) tree, which makes it easier to access and classify certain information. Thus, HTML documents are semi-structured. Semi-structured sources are the most common source on the Web because almost every website uses HTML for making the content accessible. For this reason, semi-structured sources are the main focus for information extraction (see Figure 2.2). Additionally, we can divide semi-structured sources into template-based and non-template-based semi-structured sources.

Template-based Website content is often stored in databases and the Web page is built at request time using the data from the database and a template. In this case, a template is an HTML file populated dynamically with data. Templates have several advantages. First, they are reusable, and a change to the template affects any Web pages that use the template. Second, from the perspective of information extraction, templates can be detected across several Web pages and make information extraction more reliable.

Non-template-based HTML pages that are coded by hand often do not follow a template. Similarities between the DOM trees of two HTML pages become more difficult to find, which impedes the information extraction task. Still, while many Web pages were hand coded in the early years of the Web, today most professional, data-heavy pages rely on templates (Bergman, 2001).

Structured Information Sources

Structured information sources must have a schema that describes the syntax and, to some extent, the semantics of the document (Chang et al., 2006). This schema makes it easier for machines to parse and "understand" the data. Whether or not a document is structured also depends on its content. A document might contain unstructured data in its structured parts and thereby degrade to a semi-structured information source. Examples of structured sources include databases or Extensible Markup Language (XML) files with a defined schema[2]. The most important structured format for the Web is XML and the languages that use it, such as RDF, Atom, and RSS (see Figure 2.2). We also consider the triples or quadruples of the Semantic Web to be structured data. We consider XHTML documents to be unstructured because they almost always contain large parts of unstructured data in form of plain text which makes them semi-structured documents.

2.2.2 Distribution of Document File Formats on the Web

In this thesis, we concentrate primarily on extracting textual information. In order to extract information, we need to know in which file types we can find it and how frequently those file types appear on the Web.

To the best of our knowledge, the frequency of different file types on the Web has not yet been researched. We therefore performed a simple experiment ourselves. To find the estimated number of files for each document type, we typed "* filetype:XYZ" (where XYZ is the file ending) into Google.com and took the number of indexed files as an estimate. One problem with this method is that many Web pages are generated on request by the Web server and do not even have a file ending, or they end on PHP, ASP, JSP, et cetera, which are also most often HTML files. Although that is true for any file type, we believe that most of the time, URLs without a file ending are HTML pages, which means that we have underestimated the number of HTML pages. We searched for document formats with textual content and found at least one million indexed files on Google.

In Figure 2.2, we can clearly see that (X)HTML files are the most common files on the Visible Web totaling about 87 % of all indexed files. More interestingly, we have PDF files in second place with about 2.7 % and XML in third place with just about 2.2 % of all files with a file ending.

Our conclusion from this quick analysis is that we should focus on extracting information from (X)HTML files, but might also need to look into techniques to find information in PDF and XML files. However, this analysis shows only the distribution of file types, not the value of the information within these documents.

[2]Note that a defined schema does not guarantee that the source is structured, it can still contain valid unstructured data which can make the document semi-structured.

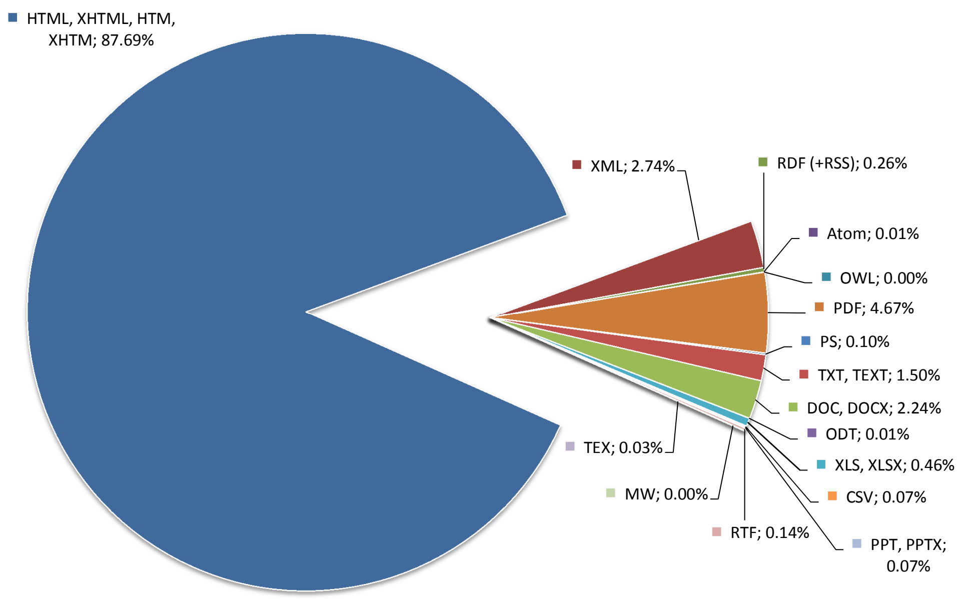


Figure 2.2: Estimated Distribution of Document File Formats on the Web as of April 2010

2.2.3 Areas on the Web

The Web can be divided into at least three major areas: the **Visible Web**, the **Deep Web**, and the **Semantic Web**. One resource on the Web can belong to multiple areas. In the literature, there is much confusion about these areas and how to describe them properly. The following paragraphs explain each of the three Web areas in more detail.

Visible Web

All resources that can be accessed using the link structure of the Web are considered to be on the Visible Web. Sometimes the Visible Web is also referred to as Indexable Web or Surface Web. The Visible Web contains at least eight billion Web pages[3] at the time of this writing. Assuming that the average Web page is about 25 kilobytes (King, 2009), the size of the Visible Web is larger than 465 terabytes.

Deep Web

The Deep Web contains all resources on the Web that cannot be reached following the link structure of the Web. More than 50 % of these resources are behind forms that provide access to topic specific databases. After querying the databases, Web pages are dynamically created dependent on the given query (Bergman, 2001). The content might also be secured and only accessible to registered users of a website, but 95 % are publicly accessible. Automatically exploring and retrieving information from the Deep Web is much more difficult than from

[3]`http://www.worldwidewebsize.com/`, last accessed on 25th of March 2012

Document Extension	**Number of Documents**
HTML, XHTML, HTM, XHTM	34,537,940,000
PDF	1,840,000,000
XML	1,080,000,000
DOC, DOCX	882,210,000
TXT, TEXT	591,120,000
XLS, XLSX	183,130,000
RDF (+RSS)	102,300,000
RTF	56,700,000
PS	38,800,000
CSV	26,900,000
PPT, PPTX	26,690,000
TEX	10,800,000
Atom	5,490,000
ODT	2,830,000
MW	1,160,000
OWL	1,050,000

Table 2.3: Estimated Distribution of Document File Formats on the Web in Absolute Numbers as of April 2010

the Visible Web. The "entry points", usually the Web forms, are sparsely distributed over the Web (Barbosa and Freire, 2007) and need to be discovered by specialized crawlers. After a set of forms has been found, it is a challenging task to submit the form with meaningful queries in order to retrieve the hidden documents. Ntoulas et al. (2005) and Madhavan et al. (2008) study the problem of automatically generating queries for these forms. Another way to access the Deep Web is through APIs. These interfaces are provided by the content owner and can be queried by a program to return data from the underlying database. The Deep Web is also referred to as Deepnet, Invisible Web, Dark Web, or Hidden Web (Olston and Najork, 2010). In 2001, it was estimated that the Deep Web is 400 to 550 times bigger than the Visible Web (Bergman, 2001), which would amount to over 93,000 terabytes of textual data.

Semantic Web

The Semantic Web, also called the Web of Data or Linked Data[4], is an initiative of the World Wide Web Consortium (Berners-Lee et al., 2001) that contains all resources on the Web that can be parsed and "understood" by machines. The Semantic Web aims to allow a new set of applications that compute and reason on the semantic information. The Semantic Web contains at least 13 billion triples[5] at the time of this writing. A triple consists of subject, predicate, and object, and can be represented in many different ways, such as RDF/XML, N3, or Turtle. Let us assume that one triple consists of 200 characters (200 bytes in ASCII). With these estimated values, we can calculate that the size of the Semantic Web is bigger than 2.3 terabytes.

To explain the differences among the three areas, we consider five dimensions shown in Figure 2.3. On all axes of the graphic, values range from one to three, except the size axis, which is measured in terabytes and is logarithmic. The five dimensions are:

1. The size in terabytes of the sum of all documents.
2. The degree of human accessibility. About 95 % of the Deep Web data is freely accessible (Bergman, 2001), but it is hidden behind forms and not accessible through multi-purpose search engines such as Google. Humans can, however, easily access the data behind the forms. The Semantic Web is primarily made for machines and is therefore not as easy to access as the Visible Web, which is primarily made to be consumed by humans.
3. The degree of human understandability. The Deep Web and the Visible Web are often encoded similarly and aim to be consumed by humans. Semantic Web documents are not encoded in a way that they could be easily understood by humans.
4. The degree of machine accessibility. The Deep Web is hidden behind forms and therefore is not easy to access for machines. The Semantic Web is primarily made for machines and is therefore easier to access than the Deep Web. The Visible Web can be almost as easily accessed by machines as by humans.
5. The degree of machine understandability. The Deep Web and the Visible Web are often encoded similarly and aim to be consumed by humans, not machines. Semantic Web documents are meant to be parsed and "understood" by machines

We have seen that the Web can be broadly divided into three major areas: the Visible Web, the Deep Web, and the Semantic Web. In this thesis, we will focus on extracting information from the Visible Web and the Semantic Web. We choose not to investigate the retrieval of documents from the Deep Web since this part of the Web is much more difficult to access. Ntoulas et al. (2005) and Madhavan et al. (2008) studied how to retrieve Web pages from the Deep Web, but we focus on extraction rather than retrieval.

[4]We treat these terms as synonyms since there are no tangible differences, Tim Berners-Lee himself said several times "Linked Data is the Semantic Web done right" (see LDOW workshop 2008 `http://blog.dbtune.org/post/2008/05/12/LDOW-and-WWW-2008`, last accessed on 25th of March 2012).

[5]`http://esw.w3.org/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics`, last accessed on 18th of June 2012

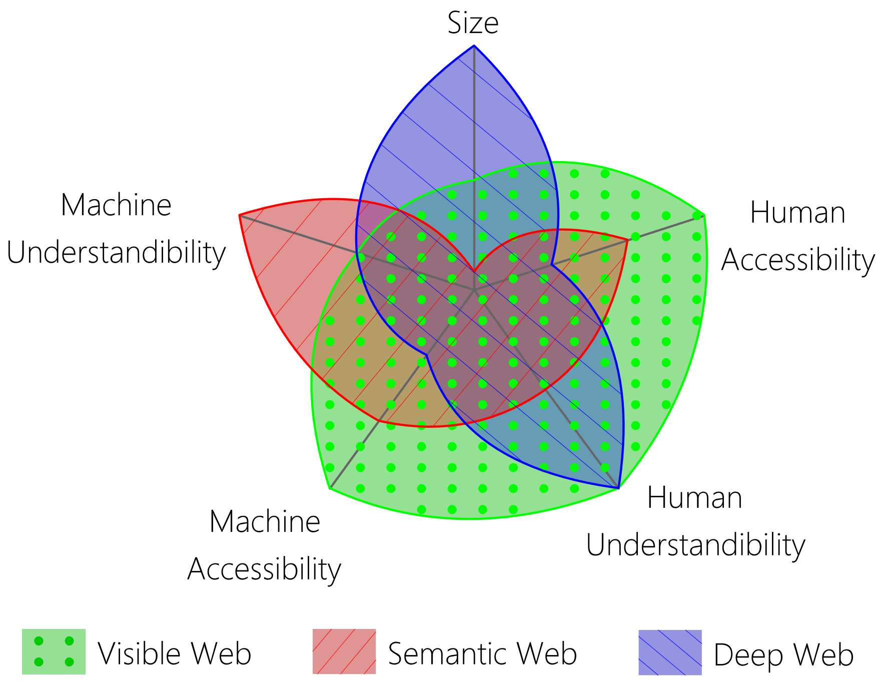


Figure 2.3: Comparison of Web Areas

2.3 Evaluation Measures and Approaches

Next, we will discuss evaluation measures and approaches that are mentioned and used in this work.

2.3.1 Performance Measurements

In information retrieval and information extraction, the most commonly used measures are precision (Equation 2.1), recall (Equation 2.2), and the harmonic mean between the two, which can be expressed in the F value (Equation 2.3). All measures have values between zero and one (often expressed in percentages), where zero is the worst and one is the best.

Figure 2.4 visualizes correct and incorrect extractions in an extraction scenario. The largest ellipse is the set of all possible extractions. The smaller black ellipse represents the set of actual extractions. A green plus (+) stands for an entity that should be extracted and a minus (−) stands for something that is not an entity and should not be extracted. "TP" are true positives, that is, correct extractions; "FP" are false positives, that is, entities that have been extracted but should not have been extracted. "FN" are false negatives, that is, entities that should have been extracted, but were not. With TP, FP, and FN we can calculate precision and recall of an extraction system in the context of a certain task. In a binary classification scenario where we have to decide for each instance whether it belongs to the target class, we have also true negative assignments ("TN"). TN are instances that were not assigned to the target concept and do in fact not belong to the class. We will need true negatives for the calculation of accuracy later on.

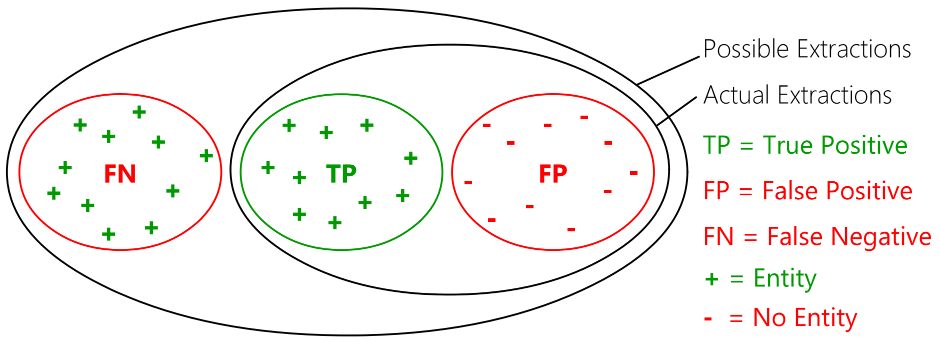


Figure 2.4: Explanation of Possible and Actual Extractions

In Web information extraction, precision measures the ratio of correct extractions to the total number of actual extractions.

$$Precision = \frac{Correct}{Actual} = \frac{TP}{TP + FP} \tag{2.1}$$

The recall is the ratio of correct extractions to the number of possible correct extractions.

$$Recall = \frac{Correct}{CorrectPossible} = \frac{TP}{TP + FN} \tag{2.2}$$

Calculating the harmonic mean of precision and recall leads to the weighted F value (van Rijsbergen, 1979):

$$F\beta = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall} = \frac{(1 + \beta^2) \times TP}{(1 + \beta^2) \times TP + \beta^2 \times FN + FP} \tag{2.3}$$

The parameter β can be used to weigh the importance of precision and recall. The higher β, the more importance is given to precision instead of recall. To facilitate the comparison of different results, the F1 value (Equation 2.4) has become the standard. For the F1 value, both precision and recall are given the same weight ($\beta = 1$), which leads to the simpler equation:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{2.4}$$

The accuracy is often used to evaluate classification tasks. To calculate the accuracy, we simply take the ratio of all correct assignments to the number of all assignments as shown in Equation 2.5.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.5}$$

Precision and recall do not take the order of the documents into account. If the order is important (for example, for search engine result pages), the precision@k and the ranked list average precision (RLAP)[6] can be used to determine the quality of the system that generated the ranked list. Precision@k is the precision of the k top ranked documents from the ranked list. Equation 2.6 shows how the precision@k is calculated, where $rel(i)$ is zero if the document at position i is not relevant and one if the document is relevant. "Relevant" is a document if it "[. . .] increases the likelihood of accomplishing the goal [. . .]" (Lindsay and Gorayska, 2002). The goal in entity extraction usually is to find entities of a given concept. All correct entity names that belong to the concept can therefore be considered relevant documents.

$$Precision@k = \frac{\sum_{i=1}^{k} rel(i)}{k} \tag{2.6}$$

The RLAP is the average for the precision for each relevant document in a ranked list. Equation 2.7 shows how this measure is calculated, where l is the ranked list of n documents and *relExpected* is the total number of relevant documents for the query.

$$RankedListAveragePrecision(l) = \frac{\sum_{k=1}^{n} Precision@k \times rel(k)}{relevantExpected} \tag{2.7}$$

Table 2.4 shows an example of a ranked list of documents and the evaluation measures at every rank k. The total number of relevant documents (*relExpected*) for this example is 5.

k	**Rel(k)**	**Relevant**	**Precision@k**	**Ranked List Average Precision**
1	1	1	$1/1 = 1.00$	$1/5 = 0.2$
2	0	1	$1/2 = 0.50$	$1/5 = 0.2$
3	1	2	$2/3 \approx 0.67$	$(1 + 2/3)/5 \approx 0.24$
4	1	3	$3/4 = 0.75$	$(1 + 2/3 + 3/4)/5 \approx 0.48$
5	1	4	$4/5 = 0.80$	$(1 + 2/3 + 3/4 + 4/5)/5 \approx 0.64$
6	1	5	$5/6 \approx 0.83$	$(1 + 2/3 + 3/4 + 4/5 + 5/6)/5 \approx 0.81$
7	0	5	$5/7 \approx 0.71$	$(1 + 2/3 + 3/4 + 4/5 + 5/6)/5 \approx 0.81$

Table 2.4: Example for Calculating Precision@k and Ranked List Average Precision

While precision@k is still independent of the order of documents, the RLAP takes the order into account. For example, if we expect four relevant documents and get the ranked list 1100, precision@4 and RLAP would be 0.5. However, if we get a ranked list 1001, the precision@4 would still be 0.5, but the RLAP would be 0.375 because the second relevant document comes later in the list.

To evaluate the performance of a system that outputs several ranked lists, the mean average precision (MAP) can be used. It is calculated by averaging the RLAP for a number of ranked lists L as shown in Equation 2.8.

[6]In information retrieval, the measure is usually called "average precision", but we use the term "average precision" in our evaluation for arithmetic means of several precision scores since we do not evaluate with ranked lists.

$$MeanAveragePrecision = \frac{\sum_l^L RankedListAveragePrecision(l)}{|L|} \tag{2.8}$$

Just as precision@k, the R-precision measures a single point in the precision-recall curve. For queries that have only r relevant documents, the R-precision makes more sense than precision@k with $k > r$. Only the top r documents are returned (R-precision equals precision@r). The R-precision is calculated as the ratio of the number of relevant returned documents to the total number of relevant documents r. The R-precision is always equal the recall and is therefore the precision-recall break-even point of the precision-recall curve (Manning et al., 2008).

2.3.2 Evaluating Named Entity Recognition

The output of NER systems is usually compared to the output of human linguists. The evaluation goal is to determine a score for the system based on this comparison. There are many different methods to calculate this score. To evaluate systems automatically, human experts have to create annotated texts with the correct solutions. For example, let us assume that a human expert created the following markup (Nadeau, 2007):

```
Unlike <PERSON>Robert</PERSON>, <PERSON>John Briggs Jr</PERSON> contacted
<ORGANIZATION>Wonderful Stockbrockers Inc</ORGANIZATION> in
<LOCATION>New York</LOCATION> and instructed them to sell all his shares in
<ORGANIZATION>Acme</ORGANIZATION>.
```

Let us also assume that an NER system created the following markup (Nadeau, 2007) for the same text:

```
<LOCATION>Unlike</LOCATION> Robert,
<ORGANIZATION>John Briggs Jr</ORGANIZATION> contacted Wonderful
<ORGANIZATION>Stockbrockers</ORGANIZATION> Inc <DATE>in New York</DATE> and
instructed them to sell all his shares in <ORGANIZATION>Acme</ORGANIZATION>.
```

The only correct match between the correct solution and the named entity recognition system output is `<ORGANIZATION>Acme</ORGANIZATION>`; all other markups are errors.

Error Types

In classification tasks, it is often possible to determine the true positives, false positives, et cetera (see 2.4), but in NER it can help to be more precise about these classes. For example, two false positives are not necessarily equally wrong. Consider a system that had to tag person names in text, and it tagged “A good start” and “Jim Carrey was” as persons. Obviously, the first occurrence is entirely wrong. The second occurrence, however, must also considered wrong, although the system only failed to find the correct right hand boundary and mistakenly tagged the word "was" too. In the previous example (see Section 2.3.2), we can see five different errors an NER system can make (Manning, 2006). The errors are shown and explained in Table 2.5 (Nadeau, 2007).

Correct Solution	**System Output**	**Error**
Unlike	`<LOCATION>` Unlike `</LOCATION>`	The system tagged an entity where none exists.
`<PERSON>`Robert`</PERSON>`	Robert	The system failed to tag an entity.
`<PERSON>` John Briggs Jr `</PERSON>`	`<ORGANIZATION>` John Briggs Jr `</ORGANIZATION>`	The system tagged the entity, but classified it incorrectly.
`<ORGANIZATION>` Wonderful Stockbrockers Inc `</ORGANIZATION>`	`<ORGANIZATION>` Stockbrockers `</ORGANIZATION>`	The system tagged the entity, but the boundaries are incorrect.
`<LOCATION>`New York`</LOCATION>`	`<DATE>`in New York`</DATE>`	The system found an entity, but classified it incorrectly and chose incorrect boundaries.

Table 2.5: Named Entity Recognition Error Types (Nadeau, 2007)

Due to the variety of combinations to weigh the error types for evaluation purposes, three main evaluation methods have evolved over the years:

Exact-match Evaluation

The exact-match evaluation is the simplest method. It does not take the different error types into account. A correct assignment must have the boundaries and the classification correct. The precision and recall are calculated as explained in Section 2.3.2. The final score for the NER system is a micro-averaged F value (MAF). The NER system from the example Section 2.3.2 would get the following scores according to Equation 2.1 and 2.2:

- $Precision = \frac{Correct}{Assigned} = \frac{1}{5} = 20\,\%$
- $Recall = \frac{Correct}{Possible} = \frac{1}{5} = 20\,\%$
- $MAF = 20\,\%$

MUC Evaluation

The MUC evaluation method takes all five errors from Table 2.5 into account and scores a system along two axes: the TYPE and the TEXT axis. If an entity was classified correctly (regardless of the boundaries), the TYPE is assigned correct. If an entity was found with the correct boundaries (regardless of its type), the TEXT is assigned correct. For both axes, three measures are used: the number of possible entities, called "POS", the number of actual assigned entities by the system, also referred to as "ACT", and the number of correct answers by the system called "COR". MUC also uses the MAF as the final score for the NER system. Like the usual F value, the micro-averaged F value is also the harmonic mean between precision and recall. Using the example from Section 2.3.2, we can calculate the MUC score for the system as follows according to Equation 2.1 and 2.2:

- $\mathit{Correct} = \mathit{COR} = 4$ (2 times TYPE correct, 2 times TEXT correct)
- $\mathit{Assigned} = \mathit{ACT} = 10$ (5 times TYPE assigned, 5 times TEXT assigned)
- $\mathit{Possible} = \mathit{POS} = 10$ (5 times TYPE, 5 times TEXT)
- $\mathit{Precision} = \frac{\mathit{Correct}}{\mathit{Assigned}} = 4/10 = 40\,\%$
- $\mathit{Recall} = \frac{\mathit{Correct}}{\mathit{Possible}} = 4/10 = 40\,\%$
- $\mathit{MAF} = 40\,\%$

ACE Evaluation

The ACE evaluation assigns weights to each entity type. For example, if an NER system correctly classifies an organization it gets one point, whereas it only gets 0.5 points for correctly tagging and classifying a person. Additionally, a cost value is set for the errors "false alarm", "missed entity", and "wrong type". The weights and costs are set for all types and their subtypes, making ACE the most customizable evaluation procedure. The final evaluation score is called Entity Detection and Recognition Value (EDR) and is calculated as 100 % minus the accumulated penalties (costs). The actual EDR for our example from Section 2.3.2 depends on the values for weights and costs. This is also a major drawback for the ACE evaluation since evaluation results might be difficult to compare. Moreover, the complex formula for the EDR complicates the analysis of errors (Marrero et al., 2009).

2.3.3 Evaluating Named Entity Discovery

When we want to evaluate the performance of an entity discovery algorithm that operates on the Web, we can calculate the precision but not the recall. The input for such an algorithm is a name of a concept, such as *Movies*. The goal of the algorithm is to find as many correct instances in the given corpus as possible. We can easily determine the precision by counting the correctly extracted entities compared to the total number of extracted entities for the given concept. We can only calculate recall, however, when we know the total amount of correct entities in the corpus. In this thesis, we use the Web as a corpus, which makes it often nearly impossible to know all the entities that can be found. While we can easily find out all entities of the concept *Country*, we struggle to find all cars, movies, actors, plants, et cetera. We therefore do not use recall as a performance measure for named entity discovery but report the number of extractions. Without the recall, we also cannot measure a harmonic mean (F value) to get a combined figure of the performance. We can, however, calculate the estimated number of correct extractions as shown in Equation 2.9, where β is the weighting parameter for the precision. The higher β, the more important the precision becomes for the final score.

$$\mathit{EstimatedCorrect} = \mathit{SampledPrecision}^{\beta} \times \mathit{NumberOfExtractions} \tag{2.9}$$

2.4 Related Systems and Knowledge Bases

We will now briefly review related information extraction systems and knowledge bases that use the Web as a source of information. This section gives a rather broad overview of work that is related to the goals of this thesis. Important parts of these works are explained in the sections of other chapters in greater detail. We will end this section with a tabular comparison of the knowledge bases.

DBpedia (Auer et al., 2007) is a project that reads factual information from semi-structured Wikipedia[7] "Infoboxes" and converts the data to Linked Data. Since the data is written by different human users, it can vary on different pages about the same concept. DBpedia cleans the data and stores it as RDF triples. These triples are then interlinked with other datasets on the Web. Over the years, DBpedia has become an important dataset that has many in- and outgoing links as shown in Figure 1.2. Since Wikipedia is the only information source of DBpedia, it also shares the same range of concepts, such as cities, musicians, books, computer games, and movies, to name a few. As of May 2010, the DBpedia dataset[8] contains over 3.6 million entities in 97 languages and over 6.2 million links to other RDF datasets (Bizer, 2011). In total, this adds up to over one billion RDF triples, about 25 % of which are extracted from the English edition of Wikipedia. The rest are extracted from other editions of Wikipedia.

Powerset[9] extracts facts from Wikipedia pages (Infoboxes and unstructured text) to allow users simple question answering functionality over these facts (called "Factz" in Powerset). If a user queries for an entity, Powerset provides information from Bing, Freebase, and Factz if there is matching information. The knowledge base is limited to English.

EntityCube[10] (Lee et al., 2010) is a research project that automatically generates summaries about people from thousands of Web pages and shows relations to other entities. EntityCube only works with entities of the people concept. It is also limited to the English Web, though a Chinese version[11] exists.

ReadTheWeb (Carlson et al., 2010a) is a project with the goal of continuously building a knowledge base from the Web by reading un- and semi-structured Web pages. This goal is very similar to ours as they use an ontology to guide the extraction process and to help assess the extractions (Mitchell et al., 2009).

[7] `http://en.wikipedia.org`, last accessed 25th of March 2012
[8] `http://wiki.dbpedia.org/Datasets`, last accessed on 25th of March 2012
[9] Formerly `http://www.powerset.com/` but it is now integrated into the Bing search engine.
[10] `http://entitycube.research.microsoft.com/`, last accessed on 25th of March 2012
[11] `http://renlifang.msra.cn/`, last accessed on 18th of June 2012

YAGO (Kasneci et al., 2008) is similar to DBpedia, except it does not primarily focus on Infoboxes, but rather on relation extraction from the unstructured text on Wikipedia pages. Due to YAGO's unifying and cleaning algorithms, information in the YAGO knowledge base reaches an average accuracy of 95 %. Suchanek (2009) expands the knowledge base using LEILA, which does not only work well on Wikipedia pages, but can extract facts and relations on any corpora of text.

Freebase (Bollacker et al., 2008) stores human knowledge in a structured way comparable to DBpedia. Freebase covers over 11 million entities, 125 million facts, and 4,000 concepts with over 7,000 attributes at a precision of about 99 % (Bollacker et al., 2008). New data is collaboratively added to the system by the community; Freebase also synchronizes with information feeds and updates information from dumps.

True Knowledge[12] is a question answering site that has a large knowledge base with entities and facts about concepts such as people, places, and products. Answers are generated by parsing the question and "calculating" the answer, that is, the machine answers, not another user. The ontology has been modeled and pre-populated by domain experts and now relies on the help of its two million users to expand the knowledge base. The knowledge base has compiled over 275 million facts[13] about over 9 million entities so far.

Wolfram | Alpha[14] is a computational knowledge engine, which means that user queries are processed and the answer is computed rather than retrieved from an existing knowledge base. Wolfram | Alpha focuses on scientific data and computation; the range of concepts is therefore not as wide as on Wikipedia. Among these are the concepts *Person*, *Country*, *City*, *Mineral*, and *Mountain*. The actual data about the entities comes from curators and is manually checked[15] so that the knowledge base is very accurate.

Kosmix (Rajaraman, 2009a,b) is a service that tries to satisfy informational searches (for example, a search for a name of an entity such as *Jim Carrey*). It categorizes the query with the Kosmix Categorization Service (Rajaraman, 2009b) and selects relevant Web pages to query. Kosmix has a large ontology with concepts and entities, information about the entities, such as facts, images, videos, and news, are searched at query time from the Deep Web through API calls. Kosmix's knowledge base automatically grows by analyzing millions of RSS entries for new entities, such as movies or artists.

Cyc (Lenat, 1995) is a manually-created knowledge base of common sense knowledge, such as "You have to be awake to eat". As the knowledge base developed, it is now possible to automatically populate the knowledge base with facts from the Web about, for example, famous people and organizations (Shah et al., 2006). However, the entities must have been entered into the knowledge base beforehand. All knowledge is given in English. Cyc has been released in two versions, "OpenCyc", a free version of the knowledge base containing about 47,000 concepts and 306,000 facts, and "ResearchCyc", a version for the research community, which comes with tools for parsing, editing, querying, and inferring knowledge.

KnowItAll (Etzioni et al., 2005) is a domain-independent, unsupervised system that automatically extracts entities and facts from the Web. The system's strength is finding new entities

[12]`http://www.trueknowledge.com/`, last accessed on 25th of March 2012
[13]`http://trueknowledge.com`, last accessed on 25th of March 2012
[14]`http://www.wolframalpha.com/`, last accessed on 25th of March 2012
[15]`http://www.wolframalpha.com/faqs9.html`, last accessed on 20th of June 2012

for a given concept using pattern learning and wrapper induction. The input for the system is a set of concepts, attributes, and relations.

TextRunner (Banko et al., 2007) goes one step further beyond the capabilities of KnowItAll because it does not require any user input. Its only input is the corpus of the Web pages, thus the approach is more scalable and easier to apply for new domains. Unlike KnowItAll, TextRunner does not use extractions from lists and is limited to the entities that can be found in free text patterns.

Alice (Banko and Etzioni, 2007) is a system that tries to learn general world knowledge, similar to that covered in Cyc, continuously from the Web. Alice builds upon TextRunner and uses its relation extraction techniques to extract information such as "`X` grows in `Y`" where `X` can be an instance of a fruit and `Y` can be an instance of a location. The system then creates a learning agenda and tries to discover more concepts and learn about them. Alice relates to this thesis as they try to learn a large knowledge base from the Web in a continuous manner. Alice's focus is, however, on "general knowledge" and topic detection instead of entity and fact extraction.

DBLife (DeRose et al., 2007) is an instance of the Cimple Project on Community Information Management. DBLife manages information about the database research community by extracting paper authors and relations between them from Web pages and bibliographies. The knowledge base can then help people in the community to find related articles and researchers. The knowledge base contains information about 407,000 entities[16].

Factual[17] is another knowledge base where users can read and submit structured data. Unlike Freebase, users may add any kind of data in tables, regardless of concepts, attributes, or entities.

Google Squared (Crow, 2010)[18] was another way to compare entities and their facts. The view is a database-like table with one entity per row and facts about the entity in the columns. Google Squared supported theoretically every concept that can be found on the Web; entities and facts were extracted from multiple sources.

Recently, Google also started building a huge graph connecting more than 200 million entities (Ulanoff, 2012). Google can then employ these connected entities to generate quick answers for search queries or propose related entities for example.

The Kylin Ontology Creator (KOG) (Wu and Weld, 2008) uses Wikipedia's Infoboxes as training data for a machine learning algorithm that uses the generated model to extract more information from the textual parts of Wikipedia pages. They therefore refine the Infobox ontology with information buried in the text. Avatar (Thathachar et al., 2006) is a rule-based information extraction system that finds relations between entities.

Other knowledge bases, such as Wikipedia, Answers[19], and START (Katz, 1997) [20], were not discussed since their data is primarily meant to be consumed by human beings and not by machines.

[16]`http://dblife.cs.wisc.edu/`, last accessed on 25th of March 2012
[17]`http://www.factual.com/`, last accessed on 25th of March 2012
[18]The service was shut down on 5th of September 2011.
[19]`http://www.answers.com/`, last accessed on 25th of March 2012
[20]`http://start.csail.mit.edu/`, last accessed on 25th of March 2012

Table 2.6 shows a comparison of the discussed knowledge bases ("K" stands for thousand and "M" for million). The table allows us to see the approximate[21] numbers of concepts, attributes, entities, and facts. These numbers are difficult to compare since some knowledge bases count entities as "topics", "terms", or even concepts. Attributes are also often relations, such as "starsIn(Actor, Movie)". We also reviewed how these knowledge bases are expanded. Possible values are "auto" for automatically using extraction techniques and "user" when it relies on input by users. If it extracts automatically, we also tried to find out which sources are used for the extraction. Our findings are approximate since some of these systems are commercial and this information is not easily available.

Knowledge Base	**#Concepts**	**#Attributes**	**#Entities**	**#Facts**	**Expansion**
ReadTheWeb	250	550	?	300 K	auto/Web
DBpedia	320	1.6 K	3.6 M	1,000 M	user/single source
Freebase	170	940	22 M	300 M	auto&user/mult.sources
True Knowledge	?	?	27.8 M	635 M	user
YAGO	250 K	92	10 M	80 M	auto/Web
TextRunner	-	-	-	7.8 M	auto/Web
(Open)Cyc	500 K	26 K	?	5 M	user

Table 2.6: Comparison of Knowledge Bases

The past paragraphs gave an overview of state-of-the-art knowledge bases and information extraction systems. While many knowledge bases provide high quality information about popular concepts, such as movies or cities, none of the knowledge bases provides sufficient methods for automatically finding new entities on the Web across a wide range of domains. Furthermore, even growing knowledge bases such as DBpedia and Freebase rely on a single source (for example Wikipedia) or users to input data. We differentiate from the introduced systems in the **approach by using and comparing multiple automatic extraction techniques**, the **focus on different areas of the Web**, and the **automatic assessment of extracted information** to improve the quality of the knowledge base.

2.5 Summary

In this chapter, we explained basic terms that are used throughout the rest of the thesis and we briefly reviewed related systems. In Section 2.1, we defined the ontological terms that we consistently use throughout the entire thesis. In Section 2.2, we outlined what kinds of sources we can find in which areas of the Web. We concluded that our focus is on extracting information from semi-structured HTML Web pages and that the Deep Web and the Semantic Web provide growing sources of valuable information. In Section 2.4, we gave a broad overview of related knowledge bases and information extraction systems. We compared the sizes and precision values of the systems and concluded that although many of them provide high quality information about popular concepts, few utilize methods for continuous, automatic

[21]Different sources have different numbers which is also due to the quick expansion of some of these knowledge bases.

expansion of their knowledge bases. In Section 2.3, we showed which evaluation measures can be applied in order to evaluate entity extraction and assessment methods.

The next chapter will present an architecture for an ontology-driven information extraction system. We will explain which entity extraction techniques we will employ, as well as how we extract statements, facts, news, question and answers, and interactive content from the Web to populate the knowledge base.

Chapter 3

Architecture

In this chapter, we introduce the architecture of the WebKnox system. The focus of the system is to extract entities and information about those entities in a way that they can be used for question answering.

Figure 3.1 presents a diagram of the components of the WebKnox system. The diagram reveals the necessary components for each step, beginning with the creation of an ontology to finally answering queries from a user. Each component will be explained in more detail in the following sections.

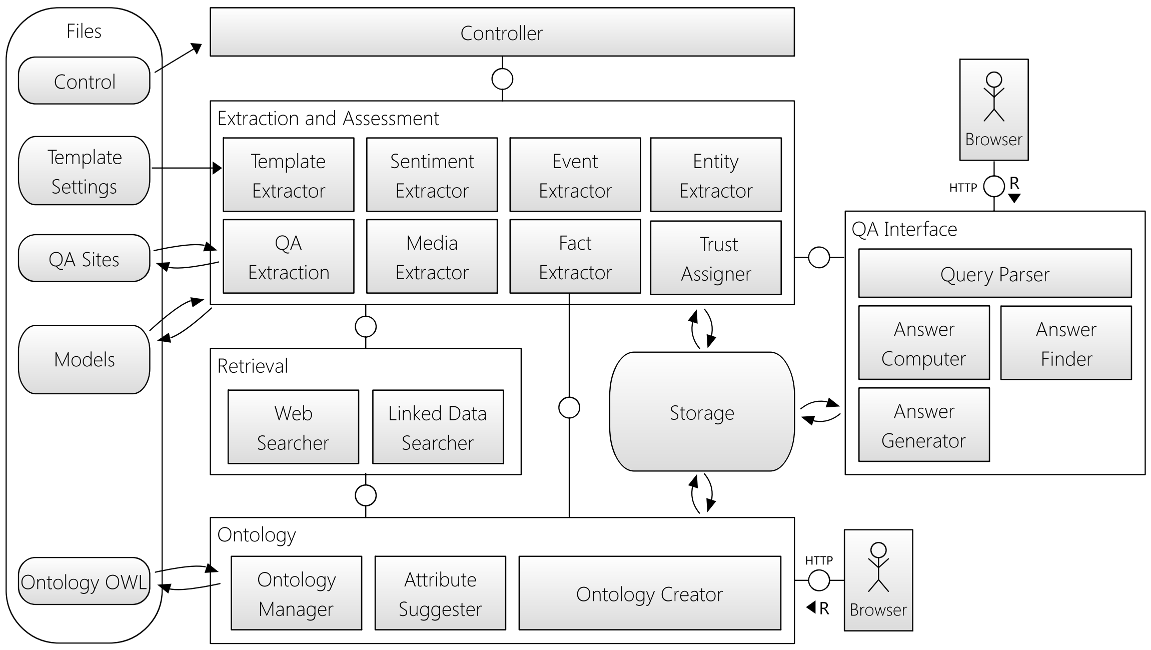


Figure 3.1: Overview of the WebKnox Architecture

While Figure 3.1 depicts the main components of the system, Figure 3.2 shows a high-level overview of the main cycle performed by the system. The main cycle consists of four basic steps that are infinitely repeated. First, all extraction components run sequentially in the extraction phase. Second, extractions are assessed; extractions that are possibly incorrect are discarded. Third, the system updates several models used in some of the extractors based on new information acquired during the extraction phase. Finally, extracted information is stored before the loop starts again.

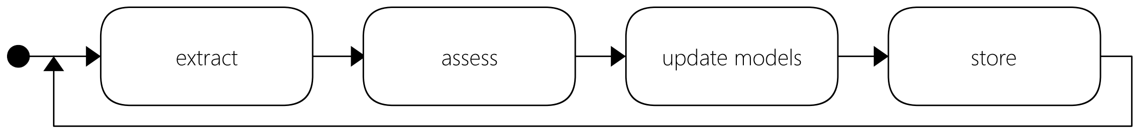


Figure 3.2: Overview of the Main Cycle of the Processes

3.1 Components

In this section, the main components of the described architecture are explained in detail.

3.1.1 Controller

The controller component is responsible for managing the main cycle shown in Figure 3.2. Each extraction component gets a time slice in which it runs its infinite extraction loop. After the time in the slice runs out, the next component continues its extraction from where it last stopped. Configurations for the length of the time slices and other meta information are stored in the Control file, which is read by the Controller component.

3.1.2 Extraction and Assessment

The extraction and assessment part of the system is observed and controlled by the Controller and consists of eight main components. All components have read and write access to the Storage and use the retrieval part of the architecture. Each component is briefly described here; a more detailed description of each component can be found in the sections dedicated to these components later.

- **Template Extractor**: The Template Extractor uses ontological information to extract sentence patterns used online to describe facts about entities in plain text format. Settings for how the extraction works are stored in the Template Settings file. Hensel (2011) has in-depth information on this topic.
- **Sentiment Extractor**: The purpose of the Sentiment Extractor is to find sentences about the entity that carry either positive or negative sentiment. See Section 8.3 for more information about this component.
- **Event Extractor**: The Event Extractor reads online news and tries to correlate entities to current events. See Section 8.2 for details about this component.

- **Entity Extractor**: The Entity Extractor discovers named entities for the given concepts in the ontology. The discovered entities are the basis for all the other extraction components. The entity extraction components and methods are a major part of this thesis and are explained in detail in Chapter 5.
- **QA Extraction**: The Question Answer Extractor uses focused crawling techniques using the QA Sites configuration file to discover and extract questions and their answers from answer-rich sites on the Web. A detailed description of the algorithm can be found in Chapter 9.
- **Media Extractor**: Entities can often be described in or linked to media such as videos, images, and interactive multimedia objects. A detailed explanation of this component can be found in Section 8.1.
- **Fact Extractor**: The Fact Extraction component uses ontological information to discover and extract facts about the entities. Chapter 7 describes five techniques to extract facts from the Web.
- **Trust Assigner**: The Trust Assigner component assesses extractions. Chapter 6 and Section 7.5 explain assessment and trust calculation in more detail.

3.1.3 Retrieval

The retrieval part contains components for accessing resources on the Web.

- **Web Searcher**: The Web Searcher connects the WebKnox system to search engines for the Visible Web, such as Google and Bing. Furthermore, it can connect the system to APIs of several services, such as Twitter and Facebook, for more information.
- **Linked Data Searcher**: The Linked Data Searcher connects WebKnox to indices of the Semantic Web, such as Sindice.

3.1.4 Ontology

WebKnox utilizes an extraction ontology that must be created by a human domain expert. The expert is, however, supported by an ontology creation tool. The ontology is read and written to an Ontology OWL file and the Storage. The tool called "Ontofly" is described in more detail in Section 7.1 and Section 10.1 and by Willner (2010), Urbansky et al. (2010), and Willner (2011).

- **Domain Expert**: The domain expert uses a Web browser to access the Ontology Creator, a tool to browse the Web and create an ontology "on the fly".
- **Attribute Suggester**: The Attribute Suggester supports the domain expert during the ontology creation. It suggests synonyms, values, datatypes, and ranges for attributes of the concepts of the ontology.

- **Ontology Creator**: The Ontology Creator is a Web application that helps the domain expert create an ontology while browsing the Web.
- **Ontology Manager**: The Ontology Manager reads and writes the ontology. It ensures that synonyms, ranges, and specific labels are stored correctly for later use.

3.1.5 Question Answering Interface

The question answer interface allows people and automated agents to query the knowledge base of WebKnox (see Chapters 9 and 10).

- **Agent**: A user or automated agent can access the interface via HTTP.
- **Query Parser**: The Query Parser analyzes the user query. It classifies the user intent and decomposes the question before finding or computing an answer.
- **Answer Computer**: The Answer Computer tries to answer the analyzed question using only the knowledge stored in WebKnox.
- **Answer Finder**: In contrast to the Answer Computer, the Answer Finder uses external sources via the retrieval part of the system to find matching answers for the given question.
- **Answer Generator**: The Answer Generator uses the extracted templates from the Template Extractor to formulate a plain text answer to a posed question using the information from the Storage (see Hensel (2011) for more information about template extraction and usage with WebKnox).

3.1.6 Storage

The Storage is where all extractions are saved and managed. In addition to the Storage, several files are used to configure and manage different components of the system. In addition to the already mentioned files, Models are stored in the file system as well. These Models have been learned in a supervised learning phase and are then used by different components for their extraction processes.

3.2 Summary

In this chapter, we presented an architecture for an entity and entity-centric information extraction system for question answering. We displayed the architecture overview before briefly describing the purpose of each of the presented components. This chapter serves as an overview of the entire system while also referring the reader to the sections in this document where the components are explained in detail.

Chapter 4

Timely Source Retrieval

One central objective of this thesis is to keep the knowledge base of entities up-to-date. To achieve this, we need to continuously retrieve new pages from the Web. The objects of interest in this chapter are feeds because they were designed for the purpose of providing the latest updates from a source. “Feeds” are a technology used in the World Wide Web to notify interested users of recent updates and changes to a website’s content. They are transferred as XML messages in one of the two standard formats, RSS and Atom, each with multiple substandards. Usually they are fetched by programs called feed readers. Feeds are used by news sites, blogs, and social media portals to announce new content to interested users. We can therefore use this technology to inform ourselves about new updates of Web pages to extract entities. For example, a movie insider website could post “J.J. Abrams started shooting Mission Impossible 6 today”. We would then need to read the feed in a timely manner and discover the new entity *Mission Impossible 6*. A small experiment on 15 news articles found on the Web (five each from BBC, CNN, and The Guardian) has shown that each news article contains about 17 entity mentions on average with four unique entities per article. We therefore believe that news Web feeds are a good source for entity discovery.

Thesis 2 in Section 1.4 states that we are able to use a moving average algorithm on the feed update intervals to outperform a fixed polling strategy and algorithms based on the post distribution of feeds. We will evaluate this thesis at the end of this chapter.

4.1 Feed Reading

In many ways, feeds replace newspapers by providing the most recent content from news sites, blogs, Web forums, and diverse social media pages. The original feed technology provides no “publish-subscribe” mechanism, which means a client application trying to receive regular updates from a feed must poll the feed’s address regularly. This has several drawbacks. First, if the polling interval is too large, updates could be missed. The naïve solution is to poll very often, increasing network traffic unnecessarily. Increased traffic is a burden on both feed consumer and provider, since it requires more powerful hardware to handle all requests. This traffic introduces more costs to server providers. For feed consumers, especially on mobile devices, it can result in additional costs if they pay per use. Even if they have a flatrate, additional polls increase the currently growing disproportion between wireless Internet traffic and its revenue (Allied Business Intelligence, 2010). On the other hand, there are instances when one wants to be notified of updates as early as possible. One example is the stock market; the reader should poll the feed right after an update occurred to be up-to-date. An additional problem is that once a user subscribes to a feed, the feed is polled for a long period of time, even if the user no longer reads it. This behavior does not occur with traditional Web pages, which are accessed only when a user is interested in their content (Sandler et al., 2005). In addition to classical feed readers, there is a growing number of systems such as Watchdog (Hu and Chou, 2009) or LocalSavvy (Liu and Birnbaum, 2008) extracting and processing information from feeds automatically, and thus putting even more load on the feed servers.

To reduce traffic, most feeds – 83 % of those used in our research – provide some means to return feed content only if it has changed since the last request. One method is the support of HTTP "LastModified", another option is the support for the "ETag" header field. Additional methods are provided by the feed standards directly. The time to live (TTL) element provides information on the time the next update will occur. To prevent polls during seldom updated times, such as on the weekend or during the night hours, some feeds support "skipHours" and "skipDays" elements. In addition, there are blogs streaming their content through an endless HTTP stream removing the load imposed by feed readers. This technique is unfortunately not adopted by many blog providers (Hurst and Maykov, 2009). Furthermore, there are patch systems, such as "pubsubhubbub" (Google, 2010), which can be seen as a moderator between feed readers and servers. All these solutions only work if both client and server support the extensions. Pubsubhubbub, for example, is only supported by 2 % of the feeds in our dataset. In fact, many clients still support only a fixed update interval, usually set to one hour by default and never changed by most users (Liu et al., 2005). It becomes obvious that inefficient polling will still be the predominant strategy for the next few years, thus motivating the development of more efficient retrieval strategies.

The goal of our research is to reduce the problems mentioned so far without changing the current technology. We show that it is possible to make a feed reader learn about the update behaviors of different feeds. An optimal strategy will minimize the number of polls to find a new item without increasing the delay between the publish time of the item and the poll significantly. We call this approach the "min delay policy".

We make the following contributions in this chapter:

- A set of algorithms to predict a feed's future update behavior based on its update history.
- An evaluation of these algorithms on a corpus of 180,000 real feeds from which we collected an update history over a span of three weeks yielding over 19 million items.
- A classification of feeds into activity classes and an analysis of class distribution over our corpus.

4.1.1 Related Work

Polling item streams with irregular updates is relevant in several research areas. We consider work in Web recrawling and previous work in feed polling.

Obtaining the most recent version of a changed Web page as early as possible is a problem found in the area of recrawling Web pages for Web search engines, which is a very active research area. Most approaches in this area model Web page updates with Poisson distributions (Tan and Mitra, 2010, Wang, 2006) and use some variation of the Post Rate approach. Since a Poisson distribution simplifies the actual update behavior of Web pages, there are other works using different probability distributions (Wolf et al., 2002), but still applying a probabilistic Post Rate update strategy. Other researchers in this area focus on the distribution of limited crawler resources (Pandey et al., 2003, Xu et al., 2010, Edwards et al., 2001, Cho and Garcia-Molina, 2003).

Over the last 10 years, the optimization of feed polling developed in two directions. The first tries to explore alternative systems for feed propagation, while the second tries to develop algorithms on top of the current feed technologies. Our approach is of the second type, but since approaches in literature often have similar ideas, we review state-of-the-art systems from both areas.

Mediator systems (Rose et al., 2007, Ramasubramanian et al., 2006, Chmielewski and Hu, 2005) are able to handle issues arising on the client side, but transfer the polling problem to the mediator system. Except for Rose et al. (2007), none of these approaches considers the problem of adaptive polling and even Rose et al. (2007) provide no solution. Sandler et al. (2005) go one step further and propose a system that changes the basic feed technology from the HTTP-based request-response paradigm to a peer-to-peer (P2P) approach. This change would be quite hard to implement, since users are used to the way feeds work today.

The adaptive feed reader algorithms described by Lee et al. (2008) and Lee and Hwang (2009) use a Post Rate update approach, calculating the probability that a new update occurs on the next day. Both depend on the semantics of the day of the week (Sunday is different from Monday) to predict the next update and both assume that a feed is run by a single person with a fixed life cycle that causes a fixed update behavior. This is sufficient for average blogs that are updated once a day (or even less often), but is insufficient for frequently updated feeds, such as feeds from digg.com where people can collaboratively post news.

Also, Liu et al. (2005) show that feed readers could save 93 % of the bandwidth if they received only deltas, and finally suggest that there are different update intervals suitable for different feeds. Unfortunately, they do not provide any algorithms.

Most similar to our approach are the algorithms described by Adam et al. (2010), Han et al. (2008), and Bright et al. (2006). All three propose adaptive feed polling algorithms to address the problems mentioned so far. Adam et al. (2010) describe an algorithm for minimizing the number of a feed's pending articles, meaning all available but not yet retrieved items. They apply static features and the feed's update history. Their system is split into two phases: a four-week training phase and an execution phase. During the training phase, the algorithm learns the polling interval it uses during the execution phase. They evaluated their system on a set of 460 feeds and achieved an improvement of 8.5 % less pending articles than with a simple round-robin approach. Unfortunately, they focus only on regularly-updated news feeds. Similar work is presented by Bright et al. (2006). However, where other approaches only consider adaptation to one update history, they propose to aggregate similar update histories and make predictions based on this aggregated information. Unfortunately, they did not specify how to find similar update histories. Their algorithm was evaluated over the course of two weeks, with seven days of a learning phase and seven days of an execution phase, on approximately 5,000 sources. It reduces requests by 47 % compared to an approach using TTL. Both algorithms need a disproportional amount of time to adapt to changing feed behavior and only Bright et al. (2006) propose a simple adaptation to feed update bursts. This adaptation, however, is insufficient if a feed changes its interval slowly over time.

Han et al. (2008) propose two goals for optimization. They introduce a "minimum delay policy" that should update each time a new item arrives and a "minimum missing policy" that should check for new items as late as possible without missing any items. They assume a constant update interval of a feed and a weight. The approach was evaluated on a dataset of 1,000 RSS feeds with updates over a time frame of six weeks and update intervals of two hours. It reduced missed items by 29 % compared to a static update policy. While their work is very similar to our approach, their focus is quite different. Similar to the works on Web recrawling, they concentrate on the assignment of limited resources and thus are only complementary to our approach.

Another approach that tries to predict the most suitable polling time for Web feed updates is proposed by Sia et al. (2007a) and Sia et al. (2007b). They try to assign polls to resources similarly to Han et al. (2008). However, their algorithms are not adaptive to changing feed behavior. Therefore, they might only work on their set of 12,000 feeds chosen from the syndic8 dataset because these feeds usually show a very periodic update behavior.

Several solutions have already been proposed to the polling problem. However, all of them have different optimization goals and focus on a specific subset of feeds. Many of them cannot adapt dynamically to changes in a feed's update behavior. Most are following the probabilistic Post Rate algorithm, which needs a large previous update history. The biggest problem for a comparative analysis, however, is the lack of a common dataset on which similar algorithms can be evaluated. We created such a dataset as described in Section 4.3.1 and made it publicly available.

4.1.2 Feed Activity Patterns

In our study of the feeds' update behaviors we defined six different patterns. These patterns cannot always be isolated and a feed might change its behavior. Nevertheless, they help us to classify feeds and develop a more sophisticated polling strategy. Figure 4.1 shows the six different activity patterns. On each time line, the dots represent the points of time of a new item in the feed. We used rule-based classification with the following features: gap between items, standard deviation of items, time to last item, and average items per day. The six patterns we identified are:

- **Zombie**: The feed is still available, but the last activity was a long time ago. We define "long time ago" as about ten times the average update interval of the feed.

- **Spontaneous**: Feeds post news every now and then without consistency. They often belong to smaller blogs with single authors showing spontaneous update behavior. The median item gap is at least one day.
- **Sliced**: These feeds exhibit a bursting behavior at certain times and inactivity at other times. We found this behavior to be typical of newspapers, which update frequently in the day time, but publish no news during the night. We classify all feeds between spontaneous and constant as sliced.
- **Constant**: These feeds post news almost constantly – day and night, even in different time zones. They often belong to large news organizations, such as BBC or Reuters. The longest item gap must be smaller than two hours, and the feed must publish at least four items each day on average.
- **Chunked**: Some feeds post several news items at exactly the same time. Therefore, multiple items have the same publish date.
- **On-the-Fly**: Some feeds generate their content on request time and all items have the current time as their publish date. This behavior makes it even harder for feed readers to estimate an update interval.

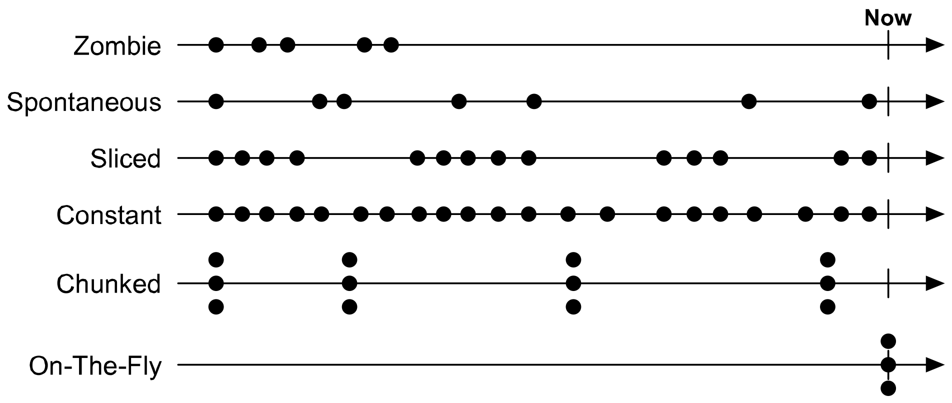


Figure 4.1: Feed Activity Patterns

It becomes obvious that the usual polling strategy of requesting the feed every hour or any other fixed time interval is not efficient. If we can classify a feed as a “zombie” we can save a significant amount of bandwidth. The same is true for “spontaneous” feeds, where most of the time there are no new items, and for “sliced” patterns, which we do not need to check at certain times, such as during the night. Figure 4.2 shows the distribution of the activity patterns in our dataset. For 94 % of the feeds (patterns “sliced”, “zombie”, and “spontaneous”) the polling strategy with a fixed interval is not sufficient. Therefore, we now propose our update strategies, which take the update behaviors of feeds into account.

4.2 Update Strategies

In this section, we describe four update strategies, three of which adapt to the feeds’ update behaviors. In the “min delay policy”, we try to read a feed as soon as it has at least one new

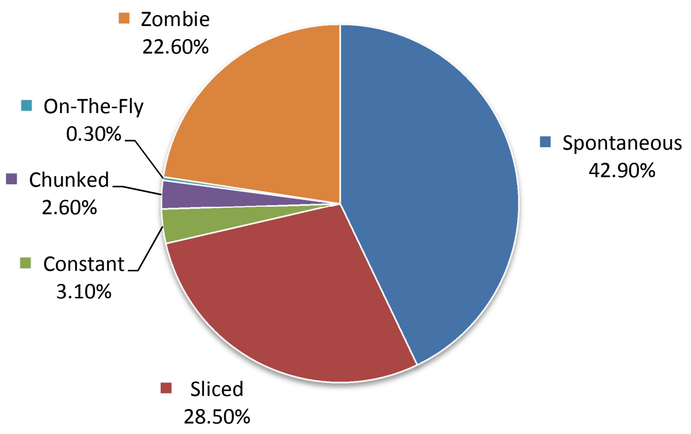


Figure 4.2: Feed Activity Pattern Distribution

item and use the following notation for modeling the problem: u is the update interval, that is, the predicted time until a new item is posted in the feed; w is the window size of the feed, that is, the number of items that can be collected at once when looking up the feed; i_n is the publish time of the n^{th} item; and p is the time of the poll. For all strategies, we limit the poll interval to [2min, 31d] to prevent constant polling or not polling at all.

4.2.1 Fix

The Fix strategy is the simplest algorithm, which polls the feed in fixed intervals. We chose one hour and one day as the two baseline intervals, Fix 1h and Fix 1d, as these intervals are reportedly common change intervals of websites (Grimes, 2008). Furthermore, once per hour is a common default update interval in feed readers.

4.2.2 Fix Learned

In this strategy, we learn the time between two new items and use it as a fixed interval for future polls. At the very first poll, we collect all publish dates for the items, calculate the average interval length as shown in Equation 4.1, and do not change this interval anymore in the future. For feeds with the "chunked" pattern, the average difference between items is likely to be zero; instead, we take the time difference from the current poll to the last entry. We also cannot calculate the average difference for "on-the-fly" patterns, so we set a fixed interval to one hour.

$$u = \frac{i_{w-1} - i_0}{w - 1} \tag{4.1}$$

4.2.3 Moving Average

The Moving Average strategy functions almost the same as Fix Learned, but updates the predicted interval u continuously. The idea is that the last intervals between new items are a good predictor for the publish time of the next item. As seen in Figure 4.2, about 28.50 % of the feeds are "sliced", that is, they change their post frequencies quite often. By continuously averaging the intervals, we can detect those frequency changes.

There are two possible observations at each time of poll p. Either there are new items in the window of the feed or not. If there are new items, we apply the moving average formula as shown in Equation 4.1. If there are no new items, we have polled too early and should increase the update interval u. Calculating the update interval with Equation 4.1 would yield no difference because we would still have the same items in the window as the last time we polled. We therefore add a "virtual item" on the timeline at the time p so that $i_{virtual} = p$ and remove the oldest one from the window. If we now calculate the update interval again, we have the chance to increase it and skip gaps of no postings in the feed. The update calculation is shown in Equation 4.2.

$$u = \frac{p - i_1}{w - 1} \tag{4.2}$$

Consider the case shown in Figure 4.3. We have a feed with the fix window size $w = 5$ that follows the "sliced" pattern, that is, there are periods of more frequent updates and periods of rare updates (period 1 to period 3). We now poll the feed for the very first time at p_0 and calculate the update interval u_0 by averaging the update intervals of the five items in our window w_0. The next poll p_1 is therefore $p_0 + u_0$. At p_1, we have five items in our window again and the new update interval will not change much since the one new item that we found and the last one in our window have similar intervals. After p_1, period 2 of less frequent updates starts. At p_2, we still have the same items in our window that we had at p_1, so calculating the new update interval u_1 using Equation 4.1 would not change anything. Since we want to increase u, we drop the oldest item in the window and replace it with the virtual item (light gray circle). The following polls are farther apart from each other because the update intervals u_1 to u_8 increase slightly due to fewer posted items. At p_9, period 3 of more frequent updates has started and we must decrease the update interval in order to lower the delay between polls and new items. We find five new items in our window and the new, smaller update interval stays about the same until p_{13} yielding only short delays between the polls and each new posted item in the feed.

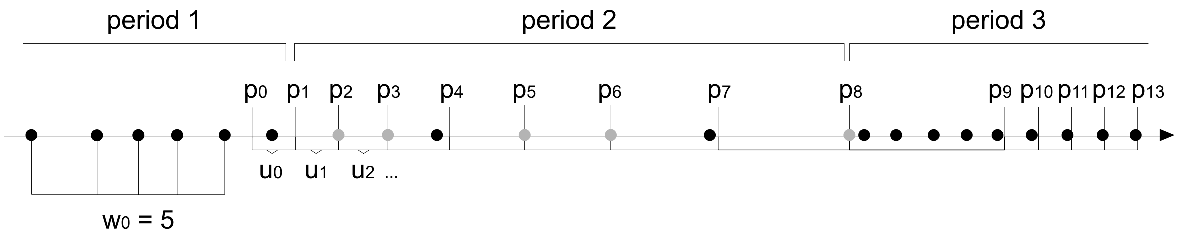


Figure 4.3: Changing Update Intervals in the Moving Average Strategy

4.2.4 Post Rate

The Post Rate strategy learns the update pattern for each feed and uses this data to predict future updates. A very similar strategy has been described by Adam et al. (2010) and Bright et al. (2006) where it is called “individual-based history adaptive strategy”. Figure 4.4 shows a sample post distribution for a feed with the hour of day on the x-axis and the probability of a new item occurring at that hour on the y-axis.

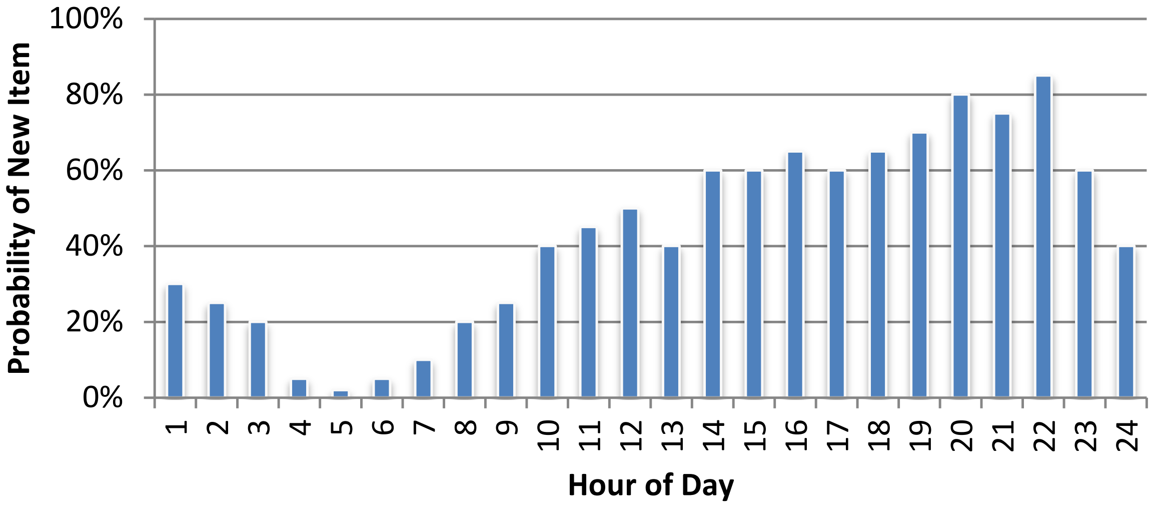


Figure 4.4: Example Post History

The goal is to find the update interval u when we can expect a new item to be posted. To find this interval, we can add the probabilities of a new item at a certain time along the timeline until the sum of probabilities is one (certainty). The sum of the number of time units that we have to move along the timeline to reach a probability of one is then our interval u. Equation 4.3 shows how to calculate the number of expected new items in a certain time frame. Our update interval u is $t_{goal} - t_{now}$ with t_{goal} being a future point in time so that the number of expected new items is one or higher.

$$newItems(t_{goal}) = \sum_{t=t_{now}}^{t_{goal}} postRate(t) \tag{4.3}$$

Consider that we poll a feed with the post distribution from Figure 4.4 at $p = 8\,a.m.$ We now sum up all post rates for the subsequent time frames after $8\,a.m.$ At $11\,a.m.$ the number of predicted items is greater than one $(0.2 + 0.25 + 0.4 + 0.45)$; therefore, in this example we should set our new update time u to three hours. While the time resolution in the given example is hours, we use a finer resolution by employing the meticulous post distributions of the feed, that is, we have 1,440 time frames with probabilities for a new item in each minute.

4.3 Evaluation

In this section, we evaluate the update strategy algorithms for the “min delay policy”, where the goal is to minimize lookups while still getting the next new item as soon as possible. A good strategy should therefore minimize the number of polls (and thereby also the network traffic) and the delay between polls and publish times of new items. We analyze each goal separately and discuss them afterwards. As baselines, we compare the Fix strategies Fix 1h and Fix 1d with the Fix Learned, Post Rate, and Moving Average strategies.

4.3.1 Dataset

To evaluate feed reading algorithms and their update strategies, we need a dataset of feeds and their items. Our most important requirement for the dataset was the diversity of the feeds in order to represent a real world snapshot of the feeds on the Web. To the best of our knowledge, no such dataset is available. For this reason, we created one and made it publicly available at the research platform Areca[1]. Liu et al. (2005) and Rose et al. (2007) used the feed collection from syndic8.com, however, we decided that this collection is not broad enough. The dataset contains about 100,000 feeds, but only every fifth comes from a different domain. Moreover, many feeds are not crawled, but rather inserted manually, skewing the distribution.

Gathering Methodology

We created keyword lists with tags and categories from Web pages, such as delicious.com, flickr.com, and dmoz.org. We used these more than 70,000 keywords and combinations of them to query Yahoo and utilized the autodiscovery[2] mechanism on the top 1,000 Web pages returned for each query. This way we gathered 240,000 feeds for a wide variety of topics from 220,000 different domains. After merging our dataset with additional feeds from syndic8.com and removing dead, empty, broken, and single item feeds, we ended up with over 180,000 feeds. It is also worth mentioning that our discovered feeds overlapped with the syndic8 dataset in only 143 URLs.

After polling each feed over the course of three weeks and storing each item along with its timestamp, title, link, number of items, and size, we had a **real** dataset of over 19 million items.

Characteristics

Despite standardization efforts, feeds are still heterogeneous in their structure and the data formats in which they are delivered. We analyzed 11 versions of RSS and Atom. By further analyzing the dataset, we discovered several feeds with up to 12,000 items per request. As this is very unusual, we discarded every feed with more than 1,000 items.

Figure 4.5 shows the feed size histogram of our dataset. Interestingly, with an average size of 55 kilobytes, a feed is more than double the size of an average Web page[3], which is about 25 kilobytes (King, 2009). This distribution shows that feeds can grow to a significant size, making an intelligent update strategy necessary. To reduce traffic, 83 % of the feeds in our dataset support either ETags or LastModified, about 13 % even support both. When a feed provides such a functionality, a feed reader that makes use of it only gets the HTTP header, which is 210 bytes on average for the feeds in the dataset. Possible savings are evaluated in Section 4.3.2.

[1]http://www.areca.co/1/Feed-Item-Dataset-TUDCS1, last accessed on 25th of March 2012

[2]Autodiscovery mechanism searches the Web page's header for explicitly stated feed URLs.

[3]Considering the HTML content alone, not all the images and additional scripts that might be referenced by the page.

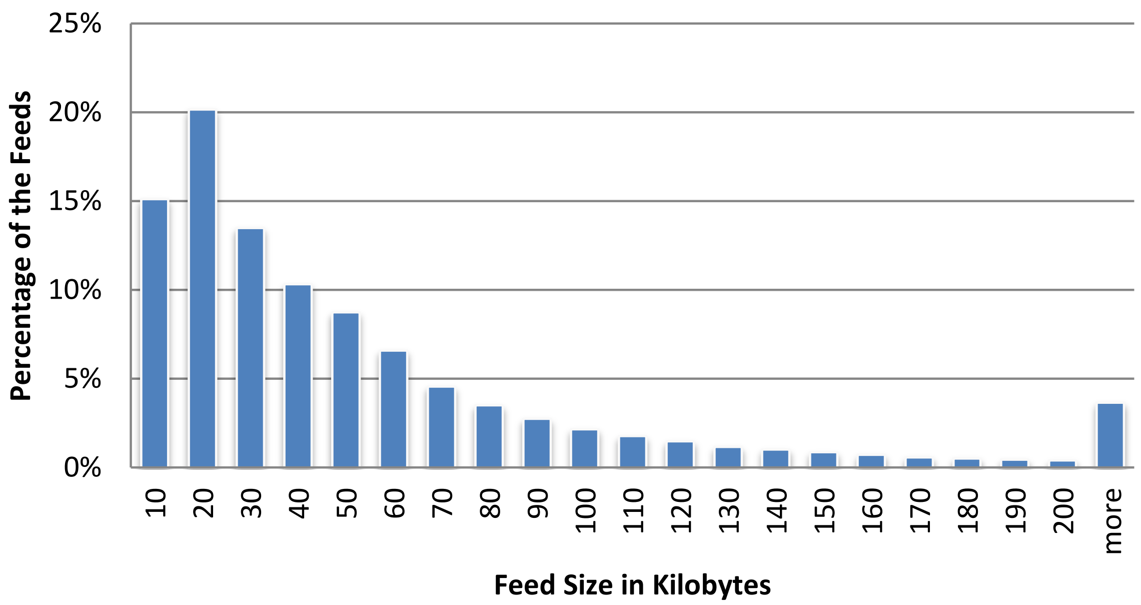


Figure 4.5: Feed Size Histogram

It is also worth mentioning that most of the feeds only had a few new items within the observation period of three weeks. While half of the dataset had only 2 new items, 10,000 feeds had more than 180 new items, and some had more than 100,000. This Zipf-distribution is shown in Figure 4.6.

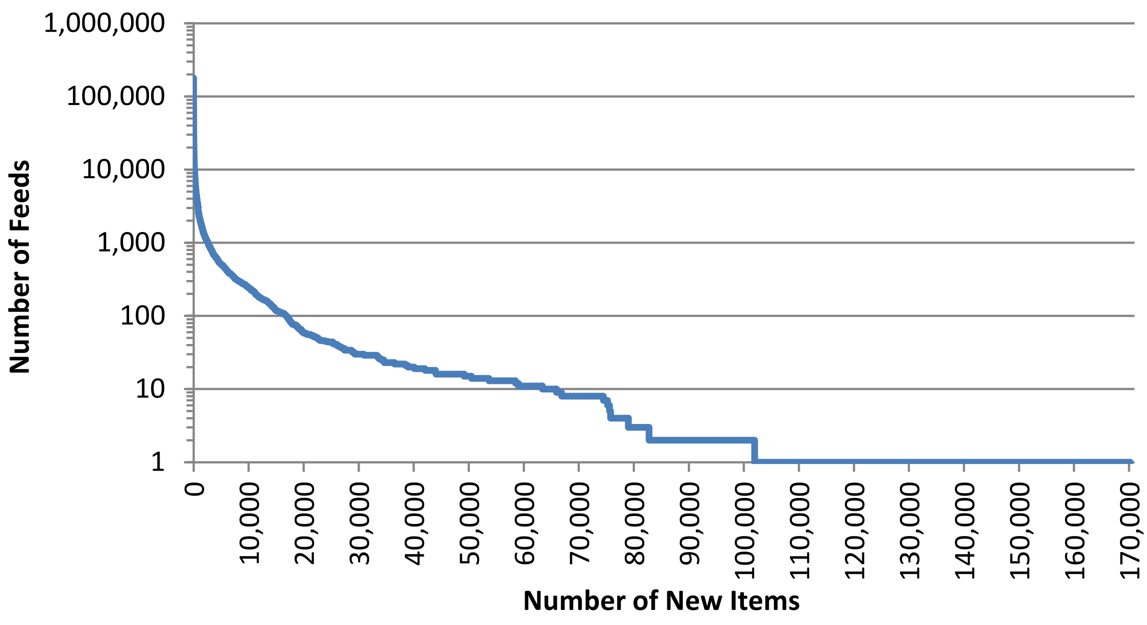


Figure 4.6: Number of Feeds versus Number of New Items

4.3.2 Network Traffic

First, we compared the network traffic of the five strategies over the course of three weeks. Figure 4.7(a) shows the total transferred amount of gigabytes for each strategy when ignoring the ETag or LastModified headers. Polling a feed once every hour, regardless of its real update behavior, leads to the highest traffic (Fix 1h).

Applying the Moving Average strategy leads to the lowest network traffic with a savings of 98.3 % in traffic volume compared to the Fix 1h strategy. Interestingly, the Fix Learned strategy performs only slightly better than Fix 1h although it takes the initial update interval into consideration. Moving Average works the same as Fix Learned, but adjusts the update interval at each new poll. We can therefore conclude that feeds change their update behaviors frequently and the update strategy benefits from picking up on the change.

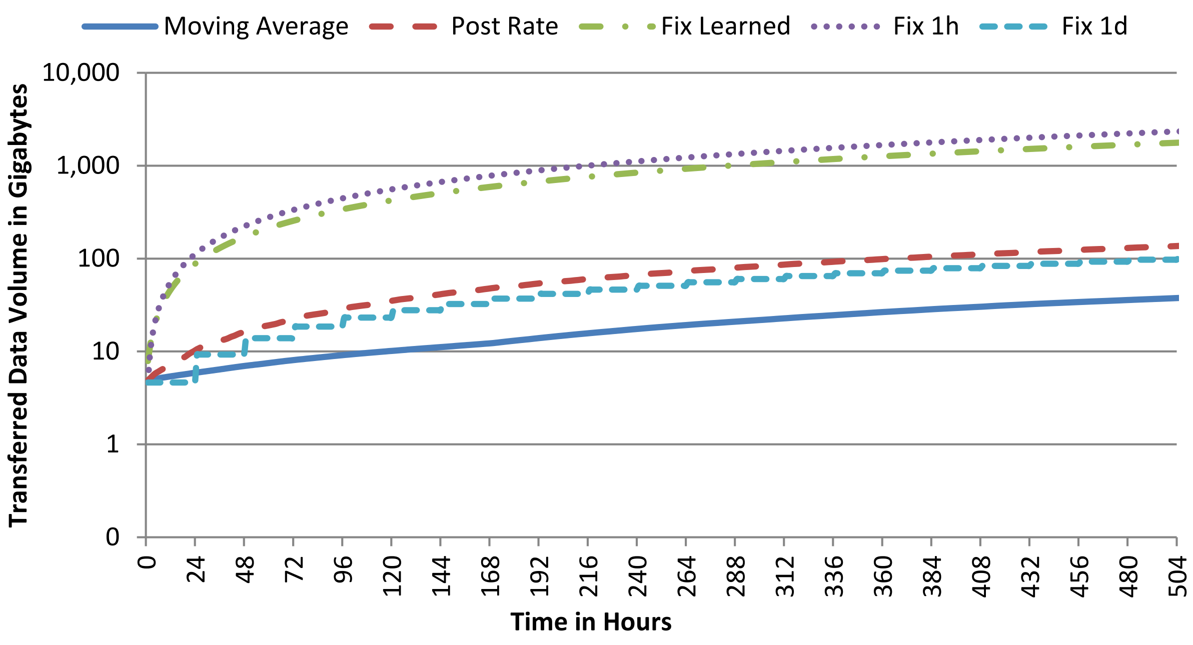
In Figure 4.7(b), we compared the network traffic again, but this time each strategy tries to reduce traffic by using the ETag and LastModified headers. 83 % of the feeds in our dataset support at least one of the two headers, which leads to an average savings in traffic volume of 95.4 % compared to not using these headers. The Fix 1h and Fix Learned strategies are still performing worst and the Moving Average is now at the level of the Post Rate strategy. The savings of Moving Average and Post Rate compared to Fix 1h are still 89.5 % and 89.8 % respectively.

4.3.3 Timeliness

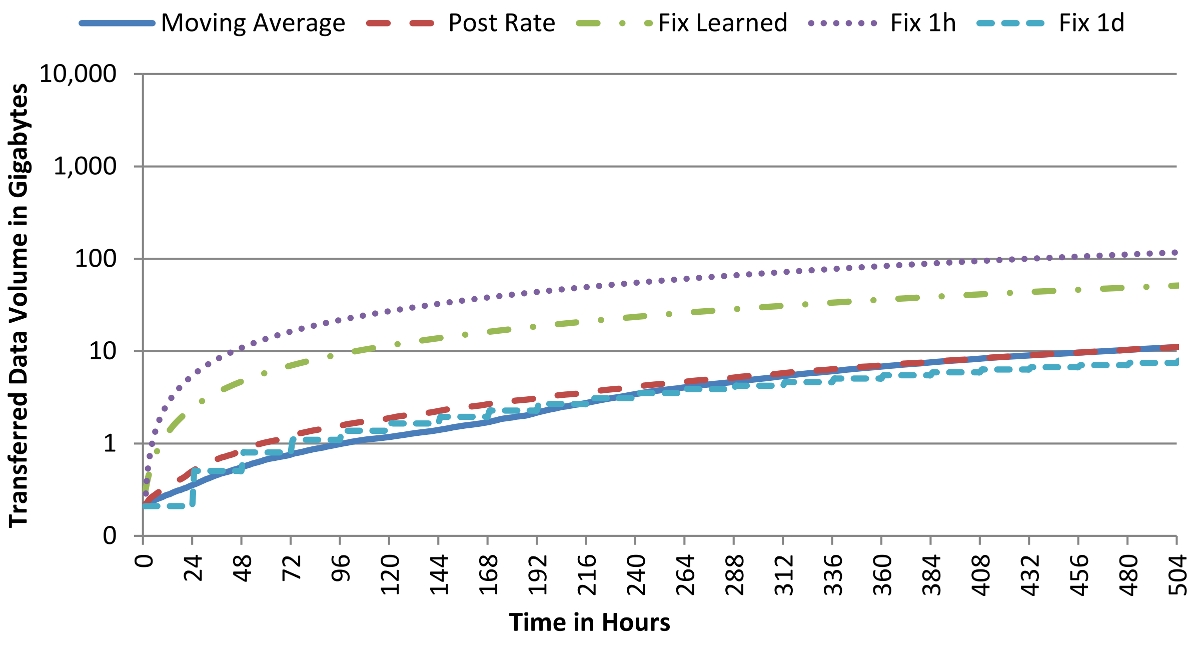
After we have compared the network traffic costs for the strategies, we now compare the delays in discovering newly posted items. Since Fix 1d polls once a day, we only selected feeds that had at least 21 updates during the creation of our 21 days dataset. Figure 4.8 shows – as expected – that Fix 1h is about 30 minutes too late on average, while Fix 1d reads each new item about 11 hours too late. Fix Learned is about 85 minutes behind on average. Post Rate learns from the complete post history of a feed and can therefore improve its update intervals. Within the first polls, it rapidly reduces its delay from seven to four hours; after that, the delay decreases more slowly. Moving Average decreases its delay at the same rate, so we are interested in how accurate they become after more polls. High average delays are caused by feeds with large update intervals.

In Figure 4.9, we had a second look at the delays, averaging over all feeds that had at least 300 updates during the creation of our dataset. This subset contained 1,541 feeds, dominated by the activity patterns Constant (54 %) and Sliced (45 %). Now it becomes obvious that the learning strategies outperform Fix 1h (and Fix 1d, not shown). Fix Learned performs best with a "constant" average delay of seven minutes, followed by Moving Average with eight minutes after 25 polls with at least one new item. Post Rate continuously learns from the growing history, but even after 300 polls, it still performs worse than Moving Average and Fix Learned.

So far, we have determined that some strategies adapt their polling intervals and decrease their average delays. As mentioned, infrequently updated feeds negatively influence the delay in minutes when averaging over all feeds.



(a) Traffic Without Using Traffic-Reducing Headers



(b) Traffic Using Traffic-Reducing Headers

Figure 4.7: Transferred Data Volume During Three Weeks Using Different Update Strategies

Figure 4.10 compares the feeds' real and predicted update intervals, using heatmaps with log_{10} scales for all axes to show how precise the strategies for a variety of update intervals are. A perfect strategy would be a straight line with $f(x) = x$. Polling with $f(x) < x$ means too often and $f(x) > x$ results in a delay. We omitted plotting the baselines Fix 1h and Fix 1d because they were horizontal lines at 1hr and 1d respectively. All figures have 31 days as the

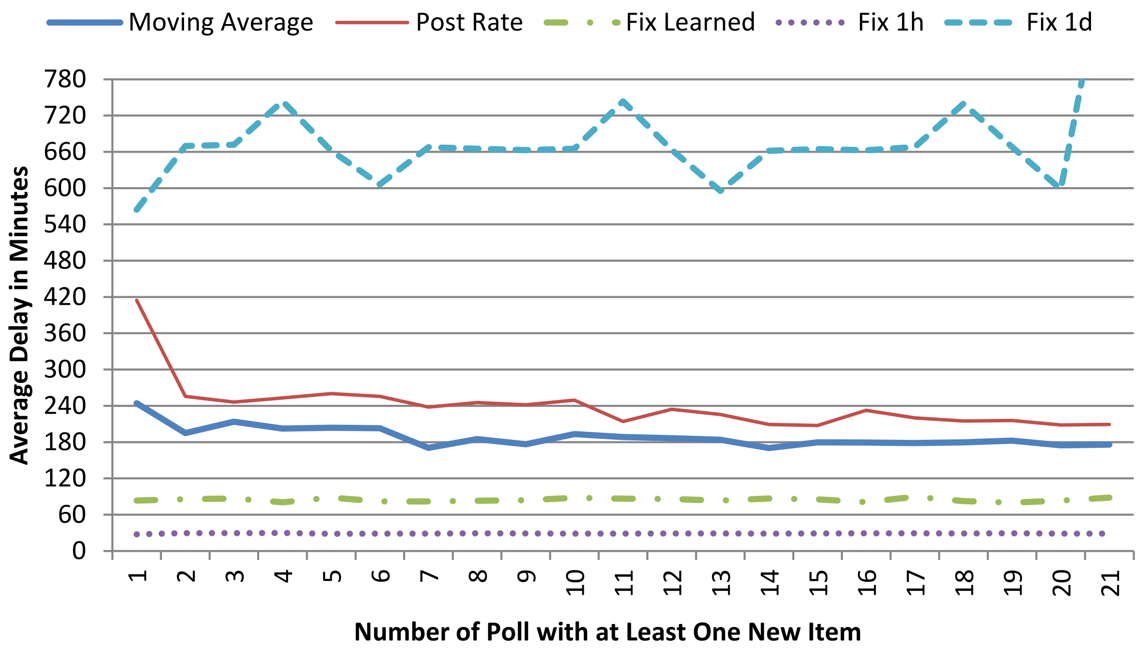


Figure 4.8: Timeliness of the Update Strategies: 21 Polls

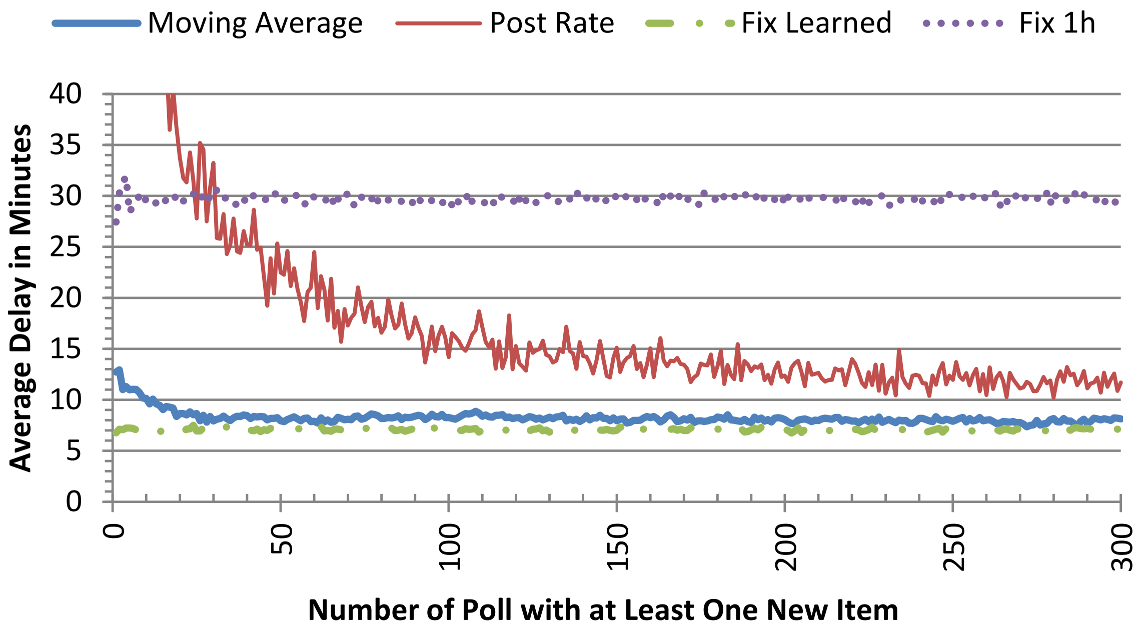


Figure 4.9: Timeliness of the Update Strategies: 300 Polls

polling intervals' upper bound (see Section 4.2). Fix Learned tends to poll too often since it uses the feed's first window to calculate the interval. The bottom line at two days is caused by "zombie" feeds that once had a high update frequency, but relapsed into inactivity. Post Rate performs better than Fix Learned. It has a lower variance and does not stay at the lower bound of two minutes. Moving Average performs best. It shows a small variance at the whole range of real update intervals. In contrast to Figures 4.8 and 4.9, the continuous

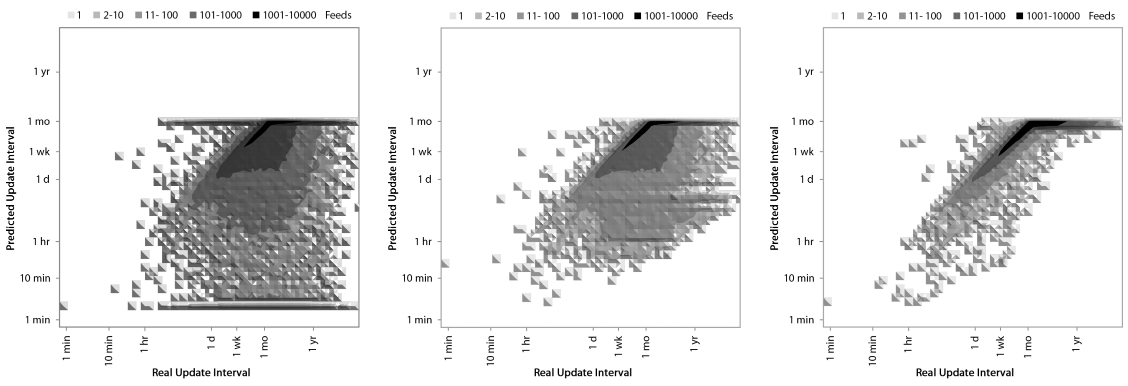


Figure 4.10: Comparing Real and Predicted Update Intervals, Left: Fix Learned, Center: Post Rate, Right: Moving Average

adapting algorithms are best when averaging over a large, broad dataset.

Table 4.1 compares the five update strategies. We compare the average delay between polls and publish times of new items, the average number of polls needed to find a new item (PPI), and the error which combines these two measures. As shown in Figure 4.6, there are few feeds with many updates, but many zombie and spontaneous feeds. We therefore used two different averaging modes, "Feeds" and "Polls". In the averaging mode "Feeds" we averaged the measures for all polls for each feed and averaged them per strategy (macro average). In the "Poll" averaging mode, we averaged over all polls of all feeds so feeds with many polls dominate the final result (micro average). We calculate the "Error" as shown in Equation 4.4. The lower the error, the better the strategy.

$$Error = Delay \times PPI \tag{4.4}$$

Mode	**Strategy**	**Delay**	**PPI**	**Error**
Feeds	Fix 1h	**30m**	222.06	**6,662**
	Fix 1d	12h:9m	9.52	6,931
	Fix Learned	54h:33m	41.90	137,139
	Post Rate	56h:51m	3.57	12,174
	MAV	74h:37m	**1.96**	8,773
Polls	Fix 1h	30m	21.02	179
	Fix 1d	12h:25m	1.91	1,423
	Fix Learned	1h:14m	7.77	570
	Post Rate	51m	5.88	300
	MAV	**4m**	**1.61**	**6**

Table 4.1: Overall Comparison of Update Strategies

We can see that although the Fix 1h strategy has the smallest delay on average, it also polls most often. The Post Rate and Moving Average strategy have similar delays, but the Post Rate strategy polls more often on average.

The best strategy can only be determined in the context of a real application. If plenty of resources are given and it is more important to get updates on time, the Fix 1h could be appropriate. We can conclude, however, that the strategy that satisfies our two requirements – a small delay and few polls – is the Moving Average strategy. Also, compared to the Post Rate strategy, which has been proposed in literature before, the Moving Average stategy is simpler since it does not require a post history to be stored.

4.4 Summary

In this chapter, we have developed and evaluated an efficient feed reading algorithm which is able to read hundreds of thousands of news feeds. This algorithm is the foundation for the extraction of entities from plain text as described in Section 5.2.4. Reichert (2012) has taken the Moving Average algorithm and developed an improved version called MAVSync, which additionally adjusts the polling intervals to synchronize with constantly updating feeds.

Chapter 5

Extraction of Entities

Finding entities is the first step in building a large knowledge base. This chapter describes state-of-the-art methods to extract named entities from the Web. First, we review related work on entity types and sources for entity extraction. Second, we explain different approaches for extracting entities before we give a detailed description of five entity extraction techniques used in WebKnox. In the last section of this chapter, we evaluate the used extraction techniques in respect to their extraction precision and the estimated number of correct extractions. This chapter searches for answers to the first thesis of this work as described in Section 1.4.

5.1 Related Work

Many researchers have dedicated their work to finding and extracting entities from text and in the past decade, there has been more focus on extracting information from semi-structured Web pages. In the following sections, we explain which entity type hierarchies have been used for entity extraction, from which types of sources different approaches have extracted entity mentions, and which extraction approaches have been employed.

5.1.1 Entity Types

Today, some NER systems recognize many types of entities, such as products, genes, or molecules, and also subtypes of the initial entity types, such as actors (people) or mountains (geographic locations). Sekine and Nobata (2004) have proposed 200 entity types and subtypes in a hierarchy, which is depicted in Figure 5.1. We can see that they also included times, such as months and days of the week, and numbers, such as monetary values and percentages, in their hierarchy.

Another hierarchy has been presented by Brunstein (2002). She used 29 entity types for a question answering task. It is rather common that high-level hierarchies (for example, *Location* instead of *City*) are used for question answering. Consider the question "Where is

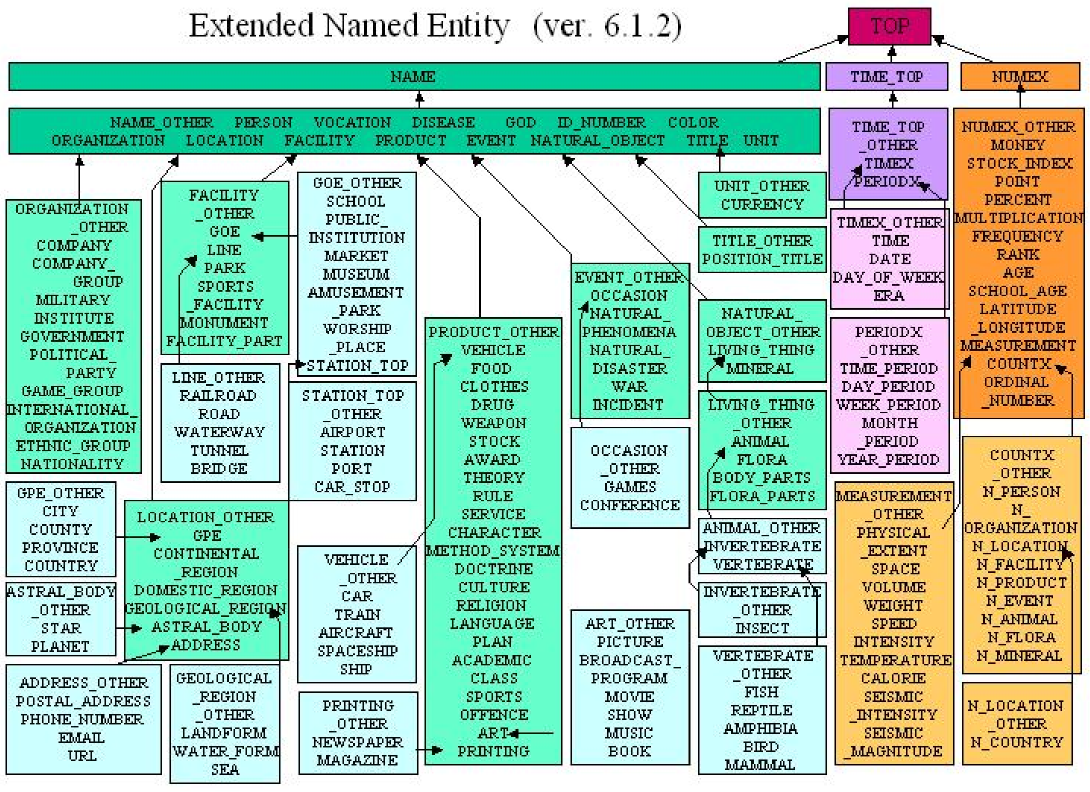

Extended Named Entity (ver. 6.1.2)
TOP
NAME
TIME_TOP
NUMEX
NAME_OTHER PERSON VOCATION DISEASE GOD ID_NUMBER COLOR
ORGANIZATION LOCATION FACILITY PRODUCT EVENT NATURAL_OBJECT TITLE UNIT
ORGANIZATION _OTHER COMPANY COMPANY_ GROUP MILITARY INSTITUTE GOVERNMENT POLITICAL_ PARTY GAME_GROUP INTERNATIONAL_ ORGANIZATION ETHNIC_GROUP NATIONALITY
FACILITY _OTHER GOE LINE PARK SPORTS _FACILITY MONUMENT FACILITY_PART
GOE_OTHER SCHOOL PUBLIC_ INSTITUTION MARKET MUSEUM AMUSEMENT _PARK WORSHIP _PLACE STATION_TOP
LINE_OTHER RAILROAD ROAD WATERWAY TUNNEL BRIDGE
STATION_TOP _OTHER AIRPORT STATION PORT CAR_STOP
PRODUCT_OTHER VEHICLE FOOD CLOTHES DRUG WEAPON STOCK AWARD THEORY RULE SERVICE CHARACTER METHOD_SYSTEM DOCTRINE CULTURE RELIGION LANGUAGE PLAN ACADEMIC CLASS SPORTS OFFENCE ART PRINTING
EVENT_OTHER OCCASION NATURAL_ PHENOMENA NATURAL_ DISASTER WAR INCIDENT
OCCASION _OTHER GAMES CONFERENCE
UNIT_OTHER CURRENCY
TITLE_OTHER POSITION_TITLE
NATURAL_ OBJECT_OTHER LIVING_THING MINERAL
LIVING_THING _OTHER ANIMAL FLORA BODY_PARTS FLORA_PARTS
ANIMAL_OTHER INVERTEBRATE VERTEBRATE
INVERTEBRATE _OTHER INSECT
VERTEBRATE _OTHER FISH REPTILE AMPHIBIA BIRD MAMMAL
GPE_OTHER CITY COUNTY PROVINCE COUNTRY
LOCATION_OTHER GPE CONTINENTAL _REGION DOMESTIC_REGION GEOLOGICAL_REGION ASTRAL_BODY ADDRESS
ASTRAL_BODY _OTHER STAR PLANET
ADDRESS_OTHER POSTAL_ADDRESS PHONE_NUMBER EMAIL URL
GEOLOGICAL _REGION _OTHER LANDFORM WATER_FORM SEA
VEHICLE _OTHER CAR TRAIN AIRCRAFT SPACESHIP SHIP
PRINTING _OTHER NEWSPAPER MAGAZINE
ART_OTHER PICTURE BROADCAST_ PROGRAM MOVIE SHOW MUSIC BOOK
TIME_TOP _OTHER TIMEX PERIODX
TIMEX_OTHER TIME DATE DAY_OF_WEEK ERA
PERIODX _OTHER TIME_PERIOD DAY_PERIOD WEEK_PERIOD MONTH _PERIOD YEAR_PERIOD
NUMEX_OTHER MONEY STOCK_INDEX POINT PERCENT MULTIPLICATION FREQUENCY RANK AGE SCHOOL_AGE LATITUDE _LONGITUDE MEASUREMENT COUNTX ORDINAL _NUMBER
MEASUREMENT _OTHER PHYSICAL _EXTENT SPACE VOLUME WEIGHT SPEED INTENSITY TEMPERATURE CALORIE SEISMIC _INTENSITY SEISMIC _MAGNITUDE
COUNTX _OTHER N_PERSON N_ ORGANIZATION N_LOCATION N_FACILITY N_PRODUCT N_EVENT N_ANIMAL N_FLORA N_MINERAL
N_LOCATION _OTHER N_COUNTRY

Figure 5.1: Hierarchy of Named Entities by Sekine and Nobata (2004)

Melbourne?". The answer is expected to be a location; it does not matter whether the entity detector can distinguish between the concepts *Country*, *Region*, and *City* in this case.

Commercial NER systems such as OpenCalais[1] and AlchemyAPI[2] also offer up to several hundred entity types.

5.1.2 Language

Most of the work on NER has been focused on English texts, but since many features are language dependent, there are approaches for several other languages, such as French (Petasis et al., 2001), Italian (Cucchiarelli and Velardi, 2001), Spanish (Sang and Meulder, 2003a), Dutch (Meulder et al., 2002), and German (Sang and Meulder, 2003b). Languages with a completely different set of characters, such as Chinese (Sun et al., 2002), Hebrew (Lemberski, 2003), Greek (Karkaletsis et al., 1999), Korean (Whitelaw and Patrick, 2003), and Arabic (Huang, 2005) were studied in literature as well. To evaluate language independent NER, Sang and Meulder (2003a) created a testbed with a German and English corpus at CoNLL 2003.

[1]`http://www.opencalais.com/documentation/calais-web-service-api/api-metadata/entity-index-and-definitions`, last accessed on 4th of May 2012

[2]`http://www.alchemyapi.com/api/entity/types.html`, last accessed on 4th of May 2012

In this work, we focus on extracting entities for the English language. Four out of five of our extraction techniques are, however, language independent and could be applied to other languages as well.

5.1.3 Source Types

Entities can be extracted from a variety of sources. This section describes related work employing extraction techniques on different sources.

Structures on Web Pages

Many Web pages use templates to present contents from databases. Therefore, pages often have a regular structure that can easily be learned. One can find many (semi-) structured formats such as HTML lists and tables on these template pages. Exploiting these structures can increase the number of entities one can extract from the Web. There are some systems (Doorenbos et al., 1997, Cohen et al., 2002, Etzioni et al., 2005, Wang and Cohen, 2007) that use techniques to extract named entities and/or relations from these structures.

Extracting Named Entities from Broadcast News

Miller et al. (1999) studied the task of extracting named entities from broadcast news. Instead of relying on written text to find new entities, they analyzed the speech of 175 hours of broadcast news and trained a statistical model on the transcripts. The difficulty here is that the transcripts often do not contain any punctuation or case information, which are often valuable features for recognizing entities. Miller et al. (1999) used the Hub4 DARPA evaluation consisting of seven concepts (*Person*, *Organization*, *Location*, *Time*, *Money*, *Date*, and *Percent*).

Extracting Named Entities from Tweets

"Tweets" are short messages (up to 140 characters) that are distributed on micro-blogging platforms, such as Twitter[3] or Tumblr[4]. The problem with tweets is that they often do not adhere to grammar rules, such as capitalization or punctuation. These features are useful to NER systems and their absence will impact the NER systems' performance. The Stanford NER system, which reaches about 90 % F1 value performance on the CoNLL 2003 shared task[5], loses about half its F1 socre and drops to about 46 % when used on tweets (Liu et al., 2011). Liu et al. (2011) have built a KNN- and CRF-based NER that works on tweets. To overcome the aforementioned problems, they use a set of orthographic and lexical features and rely heavily on the use of gazetteers. Furthermore, they employ semi-supervised learning using both recognizers to detect entities. Additionally, confidently tagged entities are fed back to the training set for the next iteration. In their experiments on about 12,000 tweets, they

[3]`http://www.twitter.com`, last accessed on 25th of March 2012

[4]`http://www.tumblr.com`, last accessed on 25th of March 2012

[5]`http://nlp.stanford.edu/projects/project-ner.shtml`, last accessed on 18th of May 2012

detected entities of the types *Person*, *Location*, *Organization*, and *Product* with an overall F1 value of about 80 %.

Extracting Named Entities from Queries

While NER is usually performed on long text documents, Guo et al. (2009) study the problem of recognizing and disambiguating named entities in user queries. According to their study, 71 % of search engine queries contain named entities and recognizing them will yield better search results. For example, they would recognize that the query "Harry Potter walkthrough" is about the video game, not the book or movie. In order to classify the queries, they learn a probabilistic model in a supervised way using a query log. They evaluated their approach on the concepts *Movie*, *Game*, *Book*, and *Music*, and reached a precision@100 of 99 %.

Similarly, Paşca (2007) shows how he uses pattern learning techniques (see Section 5.1.5) combined with seed instances to find more entities that were mentioned in user queries. In his experiment, he showed that the automatically learned extraction patterns yield an extraction precision@100 of 94 %. In a following study, Paşca and van Durme (2008) showed that query logs can be used to improve open-domain information extraction from the Web.

More recently, Jain and Pennacchiotti (2010) have shown that entity extraction from query logs can also be done in an unsupervised manner. Entity candidates are extracted by taking sequences of capitalized words or alpha-numeric characters from the query. After cleaning the candidates with a set of heuristics, each candidate is then given two scores, a Web-based "representation score" and a query-log-based "standalone score". The representation score is high if the sequence of words with its capitalization is frequently found on the Web. The standalone score is high if there are queries in the log that only contain the entity's sequence of words without any context – often only entity names are searched, increasing the chance that the candidate is indeed an entity.

Other text types

While most NER researchers train NER algorithms with features that assume correct spelling and a relatively uniform text style, Maynard et al. (2001) researched the problem of applying NER on other types of texts, such as emails, blogs, clinical notes (Wang, 2009), transcribed spoken text, Web pages, OCR output, and other text styles, such as report letters, books, lists, and texts with layouts. In this work, we will focus on correctly spelled entity names since our source is the Web and entity names usually appear more than once. We assume that the entity name is more often correctly spelled than misspelled.

5.1.4 Research Tasks

Entity extraction has been researched in multiple aspects, all pursuing slightly different goals. In this section, we review the most relevant tasks that require the extraction of entities.

Entity Extraction

We can find several terms in the literature that mean the same thing, and sometimes one term is used in different ways. We want to clarify these terms in the scope of this thesis. Under entity discovery (ED) we understand techniques to retrieve and extract entities from documents. For the scope of this thesis, ED is limited to documents that we can find on the Web. Entity recognition (ER) is the task of recognizing and disambiguating known and unknown entities in documents. Entity extraction (EE) encompasses the terms ED and ER. Figure 5.2 shows the differences and similarities of entity discovery and entity recognition.

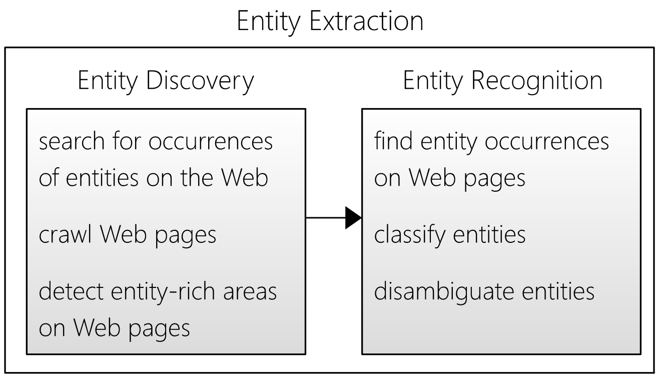

Entity Extraction
Entity Discovery
search for occurrences of entities on the Web
crawl Web pages
detect entity-rich areas on Web pages
Entity Recognition
find entity occurrences on Web pages
classify entities
disambiguate entities

Figure 5.2: Relationships between EE, ED, and ER

Entity recognition is an information extraction task that was originally defined by the Message Understanding Conference 6 (MUC-6) in 1996. The goal of the task was to identify six types of entities – *Person*, *Organization*, *Geographic Location*, *Time*, *Currency*, and *Percentage* – in texts (Grishman and Sundheim, 1996).

The input for an NER system is text and the output are named entities that were found in the given text. According to McDonald (1996), there are usually three steps that an NER system has to perform: finding entity candidates (entity delimitation), classifying candidates (entity type detection), and optionally recording them using an "Alias Network".

Entity Delimitation The first step is the entity delimitation or entity boundary detection step. In this phase, possible candidates are marked in the given text. One simple approach is to mark all sequences of capitalized words, which works well in the English language since named entities are most often capitalized. However, more sophisticated methods have also been applied. For example, Downey et al. (2007) use LEX, a statistical, n-gram-based, semi-supervised learning method to detect entity boundaries. The entity type does not have to be known in advance for their approach. Their algorithm assumes that sequences of capitalized words denote an entity. Capitalized words at the beginning of sentences need to appear sufficiently often in the text where they are not at the beginning of a sentence in order to be considered. This assumption is not valid in all languages; in German, for example, capitalization is a poor feature for entity recognition. Their approach is therefore targeted and evaluated primarily on English texts. While the authors claim that their approach works well for any entity type, they evaluate only on four relatively unambiguous types – *Actor*, *Book*, *Company*, and *Film*. Compared to state-of-the-art algorithms, they improved the performance on these entity types by about 20 % in the F1 value. Although their work seems promising, they only research the problem of detecting the entity boundaries, not the entity types. Our goal, however, is to detect unknown entities of known types.

Entity Type Detection The second step is to find the type of the entity candidates. There are three major approaches.

Lexicon-based Recognizers First, there are lexicon-based recognizers that can only detect entities that they have stored in their lexicon or gazetteer (Milne and Witten, 2008, Iacobelli et al., 2010). Often there are ambiguous entities, such as *Paris*, which is both a city (and there are many cities called "Paris") and a forename. Using lexicons is therefore not simple and matches need to be disambiguated. This approach is not of interest to us, however, since our goal is to recognize unknown entities.

Hand-crafted Rules Second, there are hand-crafted rules for NERs (McDonald, 1996, Chiticariu et al., 2010). One such rule could be "the uppercase sequence of words after the token 'Mr.' is a name". Ripper (Cohen, 1995) is a system used to create such if-then rules. Rule-based systems were mainly popular in the early days of NER research and resulted in high precision. This approach does not scale well, however, since rules need to be created by experts and must be maintained for every new entity type that should be recognized (Meulder et al., 2002). For instance, the rule for detecting person names using the prefix "Mr." has to be refined when the NER is supposed to differentiate between various kinds of people, such as politicians, actors, or professors.

Statistical Machine Learning Approaches Third, there are statistical machine learning approaches that have become increasingly popular. These approaches can broadly be divided into unsupervised and supervised machine learning. In unsupervised machine learning, the learning algorithm does not know the entity types and clusters the detected mentions. This approach often relies on patterns and lexical resources such as WordNet (Nadeau and Sekine, 2009). Downey et al. (2007) researched the task of detecting any type of entity in Web texts instead of learning particular predefined classes. They do not propose a solution for entity classification, however, but only provide a means to delimit boundaries. The most common approach is supervised machine learning in which the NER is trained on a labeled corpus of text, for example, "`<PER>`John Hiatt`</PER>` is a musician from `<LOC>`Indiana`</LOC>`". The NER builds a model from the training data and uses it to recognize the learned entity types for untagged texts later. Many machine learning techniques have been applied, including hidden Markov models (HMM) (Bikel et al., 1997), maximum entropy models (Borthwick et al., 1998), support vector machines (SVM) (Asahara and Matsumoto, 2003), decision trees (Sekine, 1998), and conditional random fields (CRF) (McCallum and Li, 2003). Also, various researchers (Wu et al., 2002, Klein et al., 2003, Kozareva et al., 2005) have shown that combining several weak classifiers yields a stronger single classifier.

Apart from these three main approaches, a number of hybrid methods have been developed, which combine rules, lexicons, and machine learning (Meulder et al., 2002).

Most approaches that we have reviewed research the problem on a very narrow domain of about four classes, which are typically *Person*, *Organization*, *Location*, and *Miscellaneous*. Some papers introduce other types, such as *Products* (Niu et al., 2003), which are still very broad and insufficient for complex applications (Fleischman and Hovy, 2002). We want to be able to recognize a wide variety of different entity types without creating and maintaining a training corpus. Downey et al. (2007) observed this problem and researched the problem of "Web NER", that is, the problem of detecting entities of any type.

Training Set Generation Supervised Machine Learning

The problem of creating labeled training was quickly recognized and researchers have since tried to ease the compilation of training data. One method to train a NER is to use a list of seed entities per entity type. The contexts around the seeds can be gathered and used for learning rules. There were several experiments with the size of the seed lists varying from only 7 per entity type (Cucerzan and Yarowsky, 1999) up to 100 (Buchholz and van den Bosch, 2000). The findings of different researchers are partly contradictory. While Collins and Singer (1999) show that as few as seven seeds are needed to create well-performing rules, Cucerzan and Yarowsky (1999) have tested the performance using between 40 and 100 seeds and found that the larger the number of seeds, the higher the F1 value due to an increased recall. Furthermore, Buchholz and van den Bosch (2000) showed that precision increased when using smaller lists with only a slight drop in recall. Whether long or short lists, seeds are a simple and fast way to train an NER, but as Meulder et al. (2002) pointed out, the recall of those systems is usually unsatisfactorily low. The recall problem can be countered using the so-called "bootstrapping" technique. In this technique, the rules learned from the contexts of the seed entities are again used to find other entities, which are then again seeds for the next learning phase (Cucerzan and Yarowsky, 1999, Niu et al., 2003, Kozareva, 2006, Whitelaw et al., 2008).

This iterative process can be performed several times to increase the number of recognizable instances; typically, the precision decreases with each iteration. Bootstrapping is also called a "semi-supervised" or "weakly-supervised" learning approach because less supervision is necessary to build the training data. Several learning strategies such as co-training (Blum and Mitchell, 1998), self-learning (Nigam and Ghani, 2000, Liu et al., 2011), using concept-based seeds (Niu et al., 2003), or domain adaptation (Wu et al., 2009) have been proposed to improve bootstrapping. However, bootstrapped approaches do not usually yield the same high performance as completely supervised methods (Kozareva, 2006).

Szarvas et al. (2007) improve the best CoNLL 2003 NER F1 value by more than 1 % by using search engine hit counts to improve the boundary detection of the token-based NER. Furthermore, they use Hearst patterns (Hearst, 1992) queries and WordNet with entity candidates to perform disambiguation between ambiguous entities.

Another approach for automatically creating training data is by automatically annotating Wikipedia texts (Kazama and Torisawa, 2007, Nothman, 2008, Balasuriya et al., 2009). Nothman et al. (2012) automatically create "silver"-standard training corpora across nine languages on Wikipedia that are comparable with manually created gold-standard corpora such as CoNLL. First, every Wikipedia article is classified and an entity type is assigned.

Links between articles become annotations with the entity type of the article that is linked. More refinement is then required to create a well-performing training corpus. For example, the first sentence of an article is often rich in proper nouns and links to other articles which makes these sentences good candidates for inclusion in the training data. Nothman et al. (2012) test their approach on a fine-grained taxonomy which only contains high-level 10 entity types, such as *Person* and *Product*. Our work differs from this approach in that we do not restrict our training set generation to Wikipedia and we test it on a finer-grained taxonomy of 17 entity types. Furthermore, we are not trying to generate a complete training corpus that can be used by every NER since that requires tagging all occurrences of entities in a text. Instead, we will show how we can create an NER that can be trained from sparsely-annotated texts.

Entity List Completion

Entity list completion (ELC), entity set expansion, or related entity finding is the task of finding and ranking related entities to a given set of "seed entities" and sometimes a description of the desired relation. For example, using the seeds *Homer Simpson*, *Ned Flanders*, and *Barney Gumble*, we would expect an ELC system to extract more characters from the TV show *The Simpsons*, such as *Bart Simpson* or *Lisa Simpson*. A given explicit description of the expected relation could, however, state that more "male cartoon character" would make the list complete.

Entity list completion is a entity extraction task that needs both entity discovery and entity recognition. Since 2007 the ELC task is regularly part of the Initiative for the Evaluation of XML Retrieval (INEX) entity ranking track, and became part of the TREC competition in the entity track[6] in 2010. We have seen that lists of entities can be used to automatically create training data for a supervised-learning-based NER which emphasizes the importance of the ELC task.

Dalvi et al. (2011) use the SEAL/Boo!Wa! system (Wang and Cohen, 2007)[7] to extract entity candidates from semi-structured Web pages that can be found in similar contexts as the seed entities. They then rank the candidates for which they found a matching URI in the billion triple index (BTC 2009 corpus). The ranking is done by comparing the type of entity candidate with the target type given in the query with the seeds. If the entity type matches the given type from DBpedia, a score of two is given, if the broader given type matches, a score of one is given, and if nothing matches, a score of zero is given. Their system reached a MAP score of only 0.0755 in the TREC 2010 ELC competition.

Google Sets (Google, 2008) is another example of a Web-based set expansion technique. Like Dalvi et al. (2011), it uses HTML patterns around the seeds to find similar entities.

Zhang and Liu (2011) research the problem of ELC in opinionated documents. Their approach is based on rule-based entity candidate extraction using Part-of-Speech tags and candidate ranking using extracted feature sets and Bayesian sets. Their main focus is finding related entity mentions in an opinionated document and not within an entire corpus. This requirement

[6]See TREC entity track website: `http://ilps.science.uva.nl/trec-entity/`, last accessed on 5th of March 2012

[7]A live demo of ELC with seal can be found at `http://boowa.com/`, last accessed on 25th of March 2012

makes the task harder since distributional similarity techniques and statistical pattern learning techniques cannot easily be applied due to a lack of data. In an evaluation of their system across ten different concepts, they achieved a precision@15 of around 78 % while Google Sets and Boo!Wa! achieved both about 69 %.

Similarly to Zhang and Liu (2011) and Bron et al. (2010), Sarmento et al. (2007) treat ELC as a ranking problem. They put all entity candidates from a given corpus into a vector space using their co-occurrence features. Features are gathered from entity listings using “and”, “or”, and commas in the corpus. As a ranking function, they use the cosine similarity measure between the seed set and the entity in the vector space.

Building upon the work of Sarmento et al. (2007), Pantel et al. (2009) use “distributional similarity” by comparing pointwise mutual information (PMI) features for vectors with the cosine similarity measure. Each term’s vector consists of pointwise mutual information scores of the term and the terms in the term’s immediate context. The more often terms appear in the same contexts, the more likely it is that they belong to the same concept.

ELC research has focused primarily on English texts so far, but as with NER, researchers have been applying ELC to other languages, such as Japanese (Sadamitsu et al., 2011) and Vietnamese (Tran et al., 2010).

Furthermore, Vyas et al. (2009) and Pantel et al. (2009) have shown that the performance of ELC systems is highly dependent on the choice of the seed entities. Randomly selected seeds and manually selected seeds influence the system’s performance by as much as 41 % in R-precision in experiments conducted by Vyas et al. (2009).

As of this writing, there are very few related works pursuing the ELC task on the Semantic Web. Lehmann et al. (2007) and Heim et al. (2009) created a system called “RelFinder”, which takes a few seeds from DBpedia (and the LOD cloud in general) and visually represents relations between them. They use a distance measure based on the connection path length between the entities – a variant of Dijkstra’s shortest path algorithm (Dijkstra, 1959). The fewer hops one has to take to arrive at another node, the more likely it is to be related. In these visualizations you can also see other related entities of the same type. Nonetheless, the ELC task is not their goal, but rather the visualization part. More related, Balog et al. (2010a) use a Semantic Web crawl to answer queries from the entity finding task. For each seed entity, given relation, and target entity type they search for entities that match that relation. Their first approach is a simple SPARQL query that returns all entities that are either subject or object in at least one triple with the source entity, and are of the target type. This approach assumes that there are explicit statements about related entities. These statements are, however, not always available. Their second approach is based on exhaustive searching, following all links from the source entity to the target type. All instances that are on the path are considered to be entities fulfilling the given relation. Since there is plenty of research on the ELC task using the Visible Web, researchers have begun employing the Semantic Web for the ELC task. They have not, however, compared the results to existing work. We will close this gap by crafting an entity extraction technique that works on interlinked data of the Web of Data, comparing it to state-of-the-art approaches that do not use the LOD cloud.

Lastly, Pantel et al. (2009) have also shown in a large empirical study that only a small number of seeds are needed for this task. This observation confirms similar experiments from the NER domain. One or two seed entities are not enough to expand entity sets effectively. Between 5 and 20 seeds yields the best performance on average. Beyond 20, the performance almost does not improve at all. We will use their observations in the evaluation of our ELC system by sampling several small and randomly selected seed sets per concept.

In conclusion, we can say that ELC is an interesting research task, which enables and improves other research fields, such as search query analysis, relation detection, automatic training set generation, and question answering. Moreover, the task has gained popularity in the community as shown by the increased amount of research papers and the recently added ELC track to TREC 2011.

5.1.5 Techniques for Extracting Entities

Now that we have explained which entity types are commonly used (see Section 5.1.1), from which types of sources (see Section 5.1.3) entities can be extracted, and in which research tasks (see Section 5.1.4) entity extraction is of importance, we review the abundance of different techniques that are used in related work.

In general, we can divide the entity extraction in approaches based on **knowledge**, **rules**, **patterns**, and **ontologies**.

Knowledge-based Approach

The knowledge-based approach uses existing knowledge of entities to recognize them in the text. We will only briefly review this approach since our goal is to extract unknown entities rather than to recognize known entities.

Wang et al. (2009) study the problem of finding misspelled or differently spelled named entities in texts by using a dictionary and the string edit distance to compare entity names. They show that using the edit distance they were able to increase the recall of the NER system.

Milne and Witten (2008) use Wikipedia as a large lexicon for entity detection, disambiguation, and linking in natural language texts, a process they call “wikification”. We should note that their definition of an entity is very different from ours. For instance, an “algorithm” should be detected and linked to the correct Wikipedia page. Following the definitions we use in this thesis, an algorithm is a concept and only instances (for example “A* search”) are considered entities. To detect entities and concepts, they employ machine learning and especially make use of the contexts around the words which are link candidates. Depending on the context, they can calculate a “link probability”, which is one indicator for the final link. Other indicators are the “relatedness” (similarity to surrounding content), “disambiguation confidence”, “generality” (more specific linking wins over more general linking), and “location and spread” (frequency and position in text). Using their machine learning approach based on bagged C4.5 trees, they were able to reach an F1 value of about 74 % on Wikipedia texts and non-Wikipedia texts.

Similarly, Iacobelli et al. (2010) create an entity detector based on Wikipedia called Wikipedia Entity Detector (WPED). Wikipedia concepts are stored in a “trie data structure” and entities are detected by comparing the words in the given text character by character to the trie path.

If the end of the word matches a leaf node in the trie, the word is considered an entity. Ambiguous entities are resolved using either popularity, that is, the more common one entity is, the higher the prior, and therefore the chance that it is correct or using proximity, that is, entities of the same kind are more likely to be found together. WPED is of course only able to detect entities that are present in the Wikipedia. The authors decided to combine their approach with the OpenCalais entity detector to improve their recall. In their research field of news articles, they were able to show that the use of a large dictionary such as Wikipedia improves the state-of-the-art NER OpenCalais by about 1 % in the F1 value.

Another problem with using dictionaries is that entities might be referred to by shortened or alternative names in the text, so that they cannot be matched exactly with the entries in the dictionary. The alternative names can be so different that normal string similarity measures and the string edit distance can no longer help. For example, a product called *Sony Cybershot DSC11* might be referred to on a Web page as *Cybershot DSC11* or just *DSC11.* Chaudhuri et al. (2009) came up with an approach to compile a list of shortened alternative names of the entity. They generated subsets of the original entity name by querying Web search engines and analyzing the context of the entity mentions.

Rule-based Approach

Rules are combined “if-then” statements that can be applied to text to extract entities. Before rules can be applied, they must either be hand-crafted or automatically learned by a machine. In this section, we also consider models that are the output of supervised machine learning as rules since, regardless of the underlying classification algorithm, one could break these models down to if-then rules.

Features Features can be understood as small information units that can be used to characterize an entity. For entity extraction, we distinguish between two kinds of features, “intrinsic” and “extrinsic” features (McDonald, 1996).

Intrinsic Features Intrinsic or internal features are built from the characters that make up the words (McDonald, 1996). Table 5.1 shows a set of these features.

Extrinsic Features In contrast to the intrinsic word-level features, the extrinsic or external document features are taken from the structure of the document and/or the corpus (McDonald, 1996). Table 5.2 (Nadeau, 2007) shows a set of this feature type.

Lists Lists are simple enumerations of words or phrases. In NER they are used interchangeably with the terms gazetteers, lexicons, and dictionaries. Most often lists enumerate named entities such as cities, person names, or company names. Lists can also be used, however, to store abbreviated or alternative versions of the same entity, for example, *GM* and *General Motors.* Furthermore, lists can store entity cues such as common prefixes or suffixes (Nadeau, 2007). For example, the prefixes “Mr.”, “Mrs.”, and “Miss” are often followed by a person name, whereas the suffixes “Inc.”, “Corp.”, and “Ltd.” are usually preceded by the name of

Feature	**Explanation**	**Examples**
Capitalization	The capitalization of the word. It can start with a capital letter, be all lowercase, all uppercase, or mixed case.	Java, yahoo, IBM, Google-Search
Patterns and case signatures	Patterns are sequences of character types (Collins, 2002).	Airbus 380 (Aaaaaa-000 or summarized as Aa-0)
Punctuation	Hyphens, ampersands, and apostrophes within the word.	I.B.M., O'Connor
Digit	Digits within the word.	W3C, 3M
Character	The character set that is used to encode the word.	Greek letters, Chinese symbols
Morphology	The prefix, suffix, singular version, stem, or a common ending of the word.	Google Inc., Yahoo Inc. (Inc. = common ending)
Part-of-Speech	The types of words in the entity name (noun, verb, adjective, et cetera).	The (article) Hangover (noun)
Function	Arbitrary functions can be applied to the entity, such as counting the tokens, constructing all n-grams (Patrick et al., 2002), or isolating non-alpha characters (Collins and Singer, 1999).	A.T.&T (..&.)

Table 5.1: Intrinsic NER Features (Nadeau, 2007)

a company (Rau, 1991). Due to the polysemy, that is, the ambiguity of terms, even complete lists are not sufficient to recognize entities reliably. Two lists for different concepts might contain the same terms; for example, the term "New York" might refer to the city, the state, a movie, or a song. Entity names can also be mistaken for common words. For example, the band called *A* might be hard to recognize in text since it could also just be the very common article. Mikheev et al. (1999) reports that in his test corpus over 20 % of entities are ambiguous.

Hand-crafted Rules One of the first rule-based systems was presented by McDonald (1996). His system called "PNF" (proper name recognition and classification facility) makes use of intrinsic and extrinsic features and works in three phases: delimit, classify, and record.

Feature	**Explanation**
Document Frequency	Entities might appear several time in the document, maybe also in different casing.
Corpus Frequency	How often the entity name appears in the corpus, also called prior probability (Whitelaw et al., 2008).
Local syntax	Position of the entity in a list, sentence, paragraph, and document.
Local context	The tokens to the left and right can give further evidence about the entity (Whitelaw et al., 2008). For example, having “Mr.” to the left of a word increases the chance for that word being a (male) person’s name.
Meta information	URI, meta keywords, document title, names of lists and figures.
Diversity	The number of occurrences across document boundaries (Whitelaw et al., 2008).
Co-occurrence	Co-occurrences of entity names, for example book titles and author names often appear together.
Informativeness	Several scores can be used as informativeness scores for single words, such as inverse document frequency (IDF), residual IDF, x^1, and mixture models (Rennie and Jaakkola, 2005).

Table 5.2: Extrinsic NER Features (Nadeau, 2007)

The delimitation phase searches for contiguous sequences of capitalized words. In the sentence “The well-known Wall Street Journal reported that Mr. Bill Gates and his Microsoft Corporation are working on a new operating system”, the character sequences “The”, “Wall Street Journal”, “Mr. Bill Gates”, and “Microsoft Corporation” would be extracted.

In the classification phase, rule checks are applied to all candidates. The first candidate “The” is classified as non-entity because the rule “single word sequences consisting solely of an article are not to be treated as names” applies. The next candidate “Wall Street Journal” is classified as an *Organization* because the keyword “Journal” is commonly used in organization names. “Mr. Bill Gates” contains the known person identifier “Mr.”, which triggers the people classification rule and labels “Bill Gates” as an instance of the concept *Person*. The last candidate “Microsoft Corporation” contains the keyword “Corporation”, which is known to often be a part of an organization’s name. Thus, the candidate is labeled as an *Organization*.

In the recording phase, each word of the classified candidates is given a role or interpretation.

`Bill Gates`, for example, matches the rule "first_name<space>last_name" so that "Bill" is assigned "first_name" and "Gates" is assigned "last_name".

Collins and Singer (1999) use a set of seven seed rules to find accurate mentions of entities. For example, they use simple containment and capitalization rules such as "if the sequence contains 'Mr.', it is a person" or "if the sequence is all uppercase, it is a company". These simple rules have proven to be very precise, although recall is low.

The advantage of hand-crafted rules is that they are easy for humans to understand. We will see in the next sections that once machine learning is involved, the "transparency" for the human is somewhat lost. Furthermore, hand-crafted rules are more precise. Consider the rule "if the instance contains 'Mr.' then it is a person". There are few instances in which this rule results in false positives. Also, although there is the initial effort of creating the rules, we do not have to label many instances as is necessary for supervised machine learning. Manually-constructed rules do have some major disadvantages. First, the recall is generally low. Consider the "Mr."-rule one more time. Person names often appear without these hints in texts and rules cannot be applied. Second, the rules are domain dependent. If the underlying taxonomy of entity types changes, rules must be manually revised. For example, what if we want to extract entities from subclasses of *Person* such as *Professor*? The "Mr."-rule would need to be revised to recognize "Professor" or "Prof.".

Rule Learning Rule learning refers to generating rules or models automatically using a supervised machine learning algorithm. There is a large body of related work concerning supervised machine learning for NER as described in Section 5.1.4. In this section, we will select only a few different approaches and explain them in more detail.

Rule Learning on Semi-structured Text Nadeau (2007) extended a rule learner from Cohen and Fan (1999) who used a rule learning algorithm called Ripper (Cohen, 1995). Nadeau (2007) uses seed entities to generate training instances with the 17 features shown in Table 5.3.

1. Tag name (nominal)
2. Text length (numeric)
3. Non-white text length (numeric)
4. Recursive text length (numeric)
5. Recursive non-white text length (numeric)
6. Depth (numeric)
7. Normalized depth (numeric)
8. Number of children (numeric)
9. Normalized number of children (numeric)
10. Number of siblings (numeric)
11. Normalized number of siblings (numeric)
12. Parent tag name (nominal)
13. Node prefix count (numeric)
14. Node suffix count (numeric)
15. Normalized node suffix count (numeric)
16. Cell row in innermost table (numeric)
17. Cell column in innermost table (numeric)

Table 5.3: 17 Features of the Rule Learner from Nadeau (2007)

An example vector that represents one (training) HTML node could be represented as follows:

```
<a,6,0,0,402,26,0.68,8,0.22,1,0.027,td,104,0.51,0,0,2,1>
```

After several training vectors have been created (positive and negative), rules can be generated. One such rule could be described in prose text as "A city name is contained in an HTML node of type `<a>`, with text length between 4 and 20 characters, in the first or second column of a table of depth 2, and with at least 20 other nodes in the page that satisfy the same rule." (Nadeau, 2007). The rules are learned using a supervised machine learning algorithm. The resulting model can then be applied to all entity candidates (for example, all terms in HTML tags) in order to classify them as entity or non-entity.

Paşca and van Durme (2008) show a weakly-supervised approach for extracting concepts, attributes for the concepts, and entities using both query logs and Web documents. Using pattern matching on queries from the 50 million query log and a set of 100 million English articles, they are able to cluster similar phrases, which are then processed to find concept labels and attributes. They manually construct 40 classes and use between 30 and 1,500 manually compiled entities for each concept. With this prior knowledge and WordNet, they are able to automatically match 37 extracted classes to their 40 classes in the gold standard. They report an accuracy of 90 % for extraction of concept names and about 80 % for entity extraction across their selected concepts. The number of entities is relatively low for many of the selected concepts. For example, they are only able to automatically extract 696 actors, 49 mountains, and 3,642 cities. The focus of their work, however, is extracting concepts and their attributes. They were able to reach a precision@10 of 70 % for automatically extracted attributes for the automatically extracted classes given only five seed attributes per class. For the concept *Accounting System*, given seed entities such as *flexcube*, *myob*, and *oracle financials*, they extracted the top ten concept attributes *overview*, *architecture*, *interview questions*, *free downloads*, *canadian version*, *passwords*, *modules*, *crystal reports*, *property management*, and *free trial*. It is questionable, however, whether these attributes can be used later in a meaningful way. The attributes *price* or *user rating* seem to make more sense than *overview*, for example.

Rule Learning on Unstructured Text: NER Supervised machine learning is the preferred technique when it comes to recognizing named entities in unstructured texts. The features for learning the model can often be used in different classifiers. Over the years, researchers have employed a wide variety of techniques to the entity recognition problem such as hidden Markov models (Bikel et al., 1997), maximum entropy models (Borthwick et al., 1998), support vector machines (Asahara and Matsumoto, 2003), decision trees (Sekine, 1998), and conditional random fields (McCallum and Li, 2003). The latter has shown to be superior to the others in several tests. It is out of the scope of this thesis to explain the machine learning approaches in greater detail. A detailed description of several machine learning algorithms is provided by Mitchell (1997).

The following example illustrates how a named entity recognizer works:

```
The well-known <ORG>Wall Street Journal</ORG> reported that Mr.
<PER>Bill Gates</PER> and his <ORG>Microsoft Corporation</ORG> are working on
a new operating system called <PRODUCT>Windows 8</PRODUCT>.
```

Note that there are different approaches and the example just shows one possibility. For supervised learning, we need labeled training data. Training data can be labeled in different formats (Urbansky et al., 2011a), but the XML annotations are common and easy to understand. The following excerpt could be part of the training data where the XML tags determine the concept and the content of the XML annotations determine the entity string. It is extremely important to note that all words not tagged in the training data are known to be non-entities. This closed-world requirement makes the process of creating labeled training data labor intensive.

The NER now creates feature vectors for the tagged entities in the unstructured text. Let us take the intrinsic features capitalization (boolean), case signature (string), part-of-speech (string), character count (numeric), and the extrinsic feature local context (string) for this example. The feature vectors representing the tagged entities in the example would be:

```
Wall Street Journal
<111,Aa-Aa-Aa,Noun-Noun-Noun,19,well-known,reported> = ORG

Bill Gates
<11,Aa-Aa,Noun-Noun,10,Mr.,and> = PER

Microsoft Corporation
<11,Aa-Aa,Noun-Noun,21,his,work> = ORG

Windows 8
<1,Aa-0,Noun-Num,9,called,.> = PRODUCT
```

We could also create feature vectors for all unlabeled tokens, such as `well-known`, and learn them as `NON-ENTITY`.

From these feature vectors, we could now learn a decision tree and finish the training phase. The saved decision tree (rules / model) can now be applied to unseen text. After delimiting the entity candidates (for example, by taking only contiguous uppercase words), we create the feature vectors and classify them using the learned decision tree.

Learning rules automatically has several advantages over crafting them by hand. First, we can let the system decide which features are important to distinguish entity types. Second, we can create a large set of features for classification, which would be difficult to maintain and understand in hand-crafted rules. Third, supervised methods scale better when we have many different entity types. Finally, the recall is generally higher than with hand-crafted rules.

On the other hand, we have disadvantages, such as the need for labeled training data, but as we have shown earlier, "bootstrapping" techniques were used to reduce this problem. Furthermore, the created model is seldom human understandable. The output is not readable, especially when creating statistical models by using machine learning algorithms, such as support vector machines.

Pattern-based Approach

Patterns are a special type of boolean rule and can be seen as a sequence of characters – if the pattern matches the rule applies. Patterns are used in a variety of different ways in related work and therefore deserve their own section. As with the rules, we distinguish between hand-crafted and automatically learned patterns. Both approaches are explained in the following sections in more detail.

Hand-Crafted Patterns A popular type of patterns are Hearst patterns (Hearst, 1992). These patterns are specific to English and can be used to detect the concept membership of an entity in unstructured text. Etzioni et al. (2005) use Hearst patterns such as the following, where `CS` is the singular name of a concept, `CP` is the plural name of a concept, and `X` is the name of an entity:

```
"CP such as X"
"such CP as X"
"CP like X"
"CP especially X"
"CP including X"
"X and other CP"
"X or other CP"
"X is a CS"
"X is the CS"
```

The KnowItAll information extraction system (Etzioni et al., 2004) initializes these patterns with a concept name, queries a search engine such as Google with the pattern, finds all occurrences of the patterns on the retrieved pages, and tries to instantiate `X` with the name of an entity. For example, if KnowItAll wants to find new instances of the concept *City*, it would query the search engine with the phrase "cities such as". It would likely retrieve Web pages with sentences such as "cities such as Los Angeles", and it would extract *Los Angeles* as a new entity of the concept *City*.

Paşca (2004) also uses the Hearst pattern, but additionally learns the concept to which the entity belongs. The pattern is therefore not instantiated with "cities such as" but only with "such as". It can then learn from the sentence "[...] other programming languages such as Java have always [...]" that "Java" is a "programming language". A similar approach was used by Evans (2003) to find new concepts in an open domain.

Pattern Learning Patterns can be constructed to find relations between entities or to extract new entities. For example, the pattern "Movies such as `LIST`" can be used to find new entities of the concept *Movie*. Constructing such patterns for every domain is too labor intensive, hence, many approaches (Talukdar et al., 2006, Agichtein et al., 2001, Brin, 1998, Etzioni et al., 2005) use pattern learning techniques that basically apply the following five steps:

1. Obtain a seed set of entities, either by using domain independent patterns (Downey et al., 2004), or by manually or automatically extracting entities from lists (Whitelaw et al., 2008, Talukdar et al., 2006).
2. Find mentions of the seed entities and create patterns which consist of the contexts of these seed entities. The context often consists of a few words from the left and right hand sides of the seed entity.
3. Find the best general patterns by filtering low frequency patterns and too short patterns. Talukdar et al. (2006) remove all patterns that can be used to extract entities from different classes. If the pattern's precision is much more important than its coverage, the learned patterns are sometimes called "sure-fire rules" (Alfonseca and Ruiz-casado, 2005).
4. Induce generated patterns to find new entities. Instead of discovering new entities, Whitelaw et al. (2008) use the learned patterns to detect "trusted mentions" of already-known entities.
5. Optionally, the newly found entities can be added to the seed set and the algorithm could start over with a larger seed set (bootstrapping).

For example, having found the entity *Jim Carrey* for the concept *Actor*, a search engine is queried with the term "Jim Carrey". On the retrieved pages, the entity appears in different patterns, such as "great performance by Jim Carrey" or "Jim Carrey played the role of his life". After ruling out bad patterns (those with few occurrences), the good patterns can be domain dependently used. For example, "`X` played the role of his life" could be a generated pattern where `X` is the placeholder for the entity name that belongs to the concept *Actor*.

We now explain these basic steps in more detail using the techniques described in SEAL (Set Expander for Any Language) (Wang and Cohen, 2007). For every seed instance, the complete prefix and complete suffix is found, that is, all characters that appear before or after the seed mention respectively. Then, all prefixes and suffixes are compared for each entity and shortened until they are the same for all seeds. The shortened wrappers can now be applied to find more instances that are encoded in the same way as the seeds.

Figure 5.3 shows an example of the wrapper construction. Figure 5.3(a) depicts an excerpt of HTML code from a Web page that mentions a number of car makes. The wrapper construction is initialized with two seed entities *Nissan* and *Toyota* (bold in (a) in Figure 5.3). Figure 5.3(b) lists the shortened prefixes and (c) lists the shortened suffixes for both seeds. These prefixes and suffixes have as many characters as both seeds have in common. For example, the second suffix ends at `[...] alt="` because for the seed *Nissan* the value of the `alt` attribute is `6` while the *Toyota* seed has the `alt` attribute value of `7`.

The prefixes and suffixes can now be applied to the given document and to this document only. A wrapper has to be constructed for each new document and is afterwards dismissed. Applying the wrappers in the example from Figure 5.3 would lead to the new entity extractions *Acura* and *acura*.

Cucerzan and Yarowsky (1999) used hierarchically smoothed trie structures for modeling contextual and morphological probabilities in a language independent manner. That is, they

(a) HTML Code Excerpt

```
<li class="acura"><a href="http://www.curryacura.com/">
<img src="/curryautogroup/images/logo-horiz-rgb-lg-dkbg.gif" alt="5"></a>
   <ul><li class="last"><a href="http://www.curryacura.com/">
       <span class="dName">Curry Acura</span>...</li></ul>
</li>
<li class="nissan"><a href="http://www.geisauto.com/nissan/">
<img src="/common/logos/nissan/logo-horiz-rgb-lg-dkbg.gif" alt="6"></a>
   <ul><li class="last"><a href="http://www.geisauto.com/nissan/">
       <span class="dName">Curry Nissan</span>...</li></ul>
</li>
<li class="toyota"><a href="http://www.geisauto.com/toyota/">
<img src="/common/logos/toyota/logo-horiz-rgb-lg-dkbg.gif" alt="7"></a>
   <ul><li class="last"><a href="http://www.geisauto.com/toyota/">
       <span class="dName">Curry Toyota</span>...</li></ul>
</li>
```

(b) Prefixes (c) Suffixes

```
 </li></ul>\n</li>\n<li class=" ENTITY "><a href="http://www.

  /">\n<img src="/common/logos/ ENTITY /logo-horiz-rgb-lg-dkbg.gif" alt="

/">\n<span class="dName">Curry  ENTITY  </span>
```

Figure 5.3: Construction of Prefixes and Suffixes with Given Seed Entities (Wang and Cohen, 2007)

built a prefix and a suffix tree with the probability for each token they encounter. In a further bootstrapping phase, they employed a discrete form of the expectation maximization algorithm on the trees.

Talukdar et al. (2006) tried to find a set of trigger words that often appear near an entity in the text and are very specific to the entity. To do so, they found a set of dominating words in all contexts around an entity. A word is dominating if it has the highest inverse document frequency weight.

Carlson et al. (2010b) showed that it is useful to couple multiple extraction techniques, assuming they make independent errors. Learning patterns from some seeds, using these patterns to extract more instances and using these new instances to learn new patterns eventually leads to a "concept drift". Concept drift means that at some point the learned patterns for the concept *Country* might be the same as for the concept *City*, such as the pattern "lived for many years in `X`", where `X` could be a country or a city. Carlson et al. (2010b) avoid that problem by mutually excluding patterns from different concepts, that is, one pattern cannot be used for multiple concepts.

Rosenfeld and Feldman (2006) applied a similar technique for relation extraction as KnowItAll, but they aimed to capture relations between two entities instead of unary relations. In their work they used patterns such as "`X` is mayor of `Y`" to find instances of mayors (`X`) and cities (`Y`). Some seed patterns must be given as an input to the system in advance.

Callan and Mitamura (2002) developed a "generate-and-test" approach called "KENE" to extract author and institution names from Web pages about conferences and workshops. Instead of learning the pattern across documents, they learn document-specific patterns by construct-

```
<body> ... <table> ... <tr> ... <td> ...
<p> <b> <font color="#000066">
INTERNATIONAL PROGRAM
COMMITTEE: </font> </b> <br>
J. Aguilar, University of the Andes, Venezuela<br>
S.E. Anderson, University of North Dakota, USA<br>
K. Araki, Hokkaido University, Japan<br>
... </td> ... </tr> ... </table> ... </body>
```

Figure 5.4: Web Page Fragment (Callan and Mitamura, 2002)

ing paths to known named entities. For example, the path `body/table/tr/td/p/br/list[1] -comma` is learned for a document given some known entities. If the path appears several time in the document it becomes a candidate. If the path is now applied to the Web page fragment from Figure 5.4, the organization names *University of Andes*, *University of North Dakota*, and *Hokkaido University* can be extracted (Callan and Mitamura, 2002).

Ontology-based Approach

We can classify information extraction approaches as "open information extraction", in which the extractor does not know what kind of relations it has to extract, or as "ontology-driven information extraction", in which the system knows an ontology to guide the extraction process. The ReadTheWeb system (Mitchell et al., 2009) is one of few existing systems using the latter approach. The ontology is not only used to know what to extract, but also to validate the extractions themselves. We focus on ontology-driven entity extraction techniques in this thesis since we believe that they yield a higher precision, which is important for a knowledge base of entities and facts.

We have now reviewed related work on entity extraction and the different tasks in this field. Our system will utilize several of the described techniques, but always within the context of ontology-driven entity extraction.

5.1.6 Summary of Related Work

We have now reviewed related work on entity extraction and found several shortcomings. First, we have shown that there are many general extraction approaches which can be applied to entity extraction but a detailed analysis and comparison of the different approaches does not yet exist. Second, many researchers focus only on a small number of concepts (typically four) which is often not enough for real world applications[8]. Third, the problem of creating training data for machine learning algorithms has been researched more intensively in the recent past but is still an open problem, especially for larger ontologies of concepts. Finally, there are still untapped sources for entity extraction, such as the Semantic Web.

[8]Commercial services such as AlchemyAPI and Open Calais offer larger ontologies. We assume this is due to the demand in the industry.

In the next sections, we will describe five entity extraction techniques that solve parts of the open problems mentioned.

5.2 Extraction Techniques

The following sections describe five extraction techniques used by WebKnox. Our contributions are the adaption and modification of two existing entity extraction algorithms and the development of three novel techniques. The last section of this chapter evaluates the described approaches.

Figure 5.5 depicts the entity extraction cycle, which runs in an infinite loop until the time slot for entity extraction expires and the next extraction component takes over. First, the ontology is loaded. Second, for each selected concept, five entity extraction techniques are used in sequence. Finally, the results are stored for later assessment.

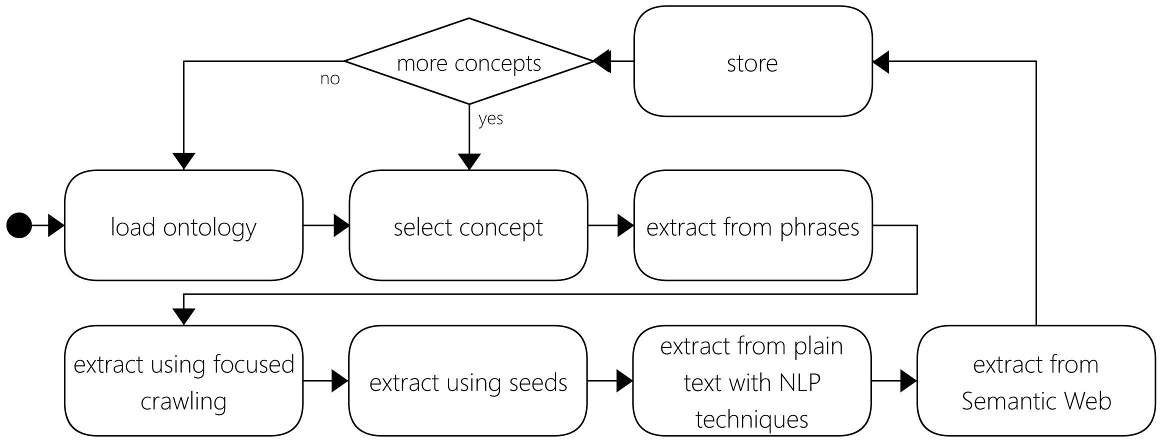

more concepts
no
yes
store
load ontology
select concept
extract from phrases
extract using focused crawling
extract using seeds
extract from plain text with NLP techniques
extract from Semantic Web

Figure 5.5: Overview of the Processes in the Entity Extraction Cycle

5.2.1 Extraction Using Phrase Patterns

The Phrase Extraction technique borrows the basic idea from the KnowItAll system (Etzioni et al., 2005) and queries a search engine with phrases that are likely to link a concept to several entities of that concept.

Queries

The following queries are used by the Phrase Extraction technique where `CP` is the plural name of a concept. The quotes are part of the query, that is, only phrase matches are sought.

```
"CP such as"
"CP like"
```

```
"CP including"
"CP especially"
```

Extraction

For each concept, all queries are instantiated with the concept name and sent to a search engine. WebKnox then searches the phrases on the returned pages for each query and proper nouns after each phrase are extracted as entities. For example, for the concept *Country* the first query would be instantiated with "countries such as" and might return a Web page that states the phrase "[...] countries such as Australia, New Zealand, and Fiji are known for their strength in rugby". The countries after the phrase are proper noun entities and can be extracted as instances of the concept *Country*.

5.2.2 Focused Crawling Extraction

The Focused Crawl Extraction technique queries a search engine with generic queries targeted toward finding pages with lists of entities. The focused crawler tries to detect a list on the page and also searches for signs of pagination. Pagination is often used to limit the results to one page and make it possible for the website visitor to view the data page by page. Figure 5.6(a) to (d) show such typical paginations. If pagination is detected, the focused crawler starts on the first page, tries to detect a list, and then extracts entities from the list. If a list is detected and entities are extracted, the focused crawler moves on to the next page in the pagination. The process repeats until all pagination links are crawled. If no pagination is found, the focused crawler tries to detect a list on the Web page and extracts entities from that list.

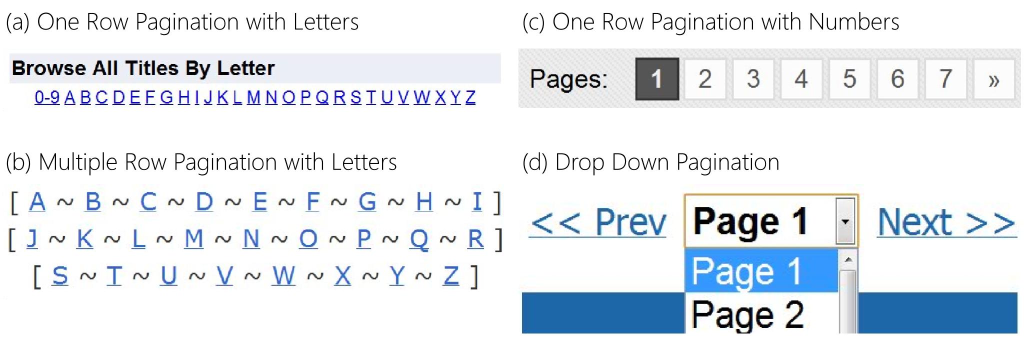

(a) One Row Pagination with Letters
Browse All Titles By Letter
0-9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
(c) One Row Pagination with Numbers
Pages: 1 2 3 4 5 6 7 »
(b) Multiple Row Pagination with Letters
[A ~ B ~ C ~ D ~ E ~ F ~ G ~ H ~ I]
[J ~ K ~ L ~ M ~ N ~ O ~ P ~ Q ~ R]
[S ~ T ~ U ~ V ~ W ~ X ~ Y ~ Z]
(d) Drop Down Pagination
<< Prev
Page 1
Next >>
Page 1
Page 2

Figure 5.6: Examples of Pagination on Websites with Letters (a) and (b), Numbers (c), and Drop Down Menu Texts (d)

The following sections explain which queries are used to find pages with lists of entities, how these lists and pagination links are detected, and how the extraction is performed.

Queries

The focused crawler processes pages that are retrieved with the following queries, where `CS` is the name of the concept and `CP` is the plural of the concept name:

```
"list of CP"
"CS list"
"index of CP"
"CS index"
"browse CP"
CS A-Z
```

These queries aim to find pages that explicitly state to have a "list" (or "index") of the desired concept on it. The "browse" keyword aims to find pages that allow a user to browse entities of a concept. A user can find information on a website in two ways: by searching for it using a website-specific search functionality, or to browse for it, that is, to follow links on the Web pages until the sought information is found. Sometimes websites explicitly use the term "browse" to indicate to the user that he can try to stumble upon the information he wants. These websites often use pagination to provide a page-by-page view of information coming from a database.

List Detection

List detection aims to find the XPath that points to all entities of a list. This is a complicated task since lists can be encoded in a variety of ways, and it is often not even clear what should be considered a list and what should not. For the list detection algorithm, a list must have the following features:

1. The entries of the list are structured in a very similar way, that is, the XPaths that address the entries are the same (only indices on XPaths differ).
2. There are at least 10 entries in the list.
3. The entries of the list are uniform, that is, they have a similar format (for example, word length or capitalization is consistent among the entries).
4. The list is in the content area of the Web page. Enumerations in the Web page's header, navigation, or footer are not considered to hold entities of interest.

The correct path can rarely be found by looking solely at the page's DOM tree. For this reason, the content of a list is also analyzed in order to find the sought list. The list detection algorithm explained here makes use of many heuristics, which were determined by analyzing a wide variety of list pages.

Content Tags Entities are expected to be in one of the following tags:

`LI, TD, H2, H3, H4, H5, H6, A, I, DIV, STRONG, SPAN, OPTION`

The list detector does not construct an XPath to every tag that can be used in HTML since some tags are not designed to contain text content, such as `TABLE`, `TR`, and `SELECT`.

The list detector creates a simple XPath to all occurrences of all the tags listed above. XPath is flexible and allows us to address one and the same node in different ways; the list detector creates each XPath in the simplest way by going from the addressed node to the parent node until it reaches the root node.

Highest Count XPath Every constructed XPath addresses exactly one node. To find the XPath that addresses all entities, that is, multiple nodes, the indices of the nodes in the XPath are removed (only the index of the `TABLE` tag remains in order to handle pages with multiple tables). Figure 5.7 visualizes this step. In (a) only one `li` node is addressed, while in (b) the indices of the XPath are removed and all `li` nodes are addressed. The path addressed by the XPath in (a) and (b) is marked by green rectangles around the nodes. An XPath with its indices removed is called a “stripped XPath”.

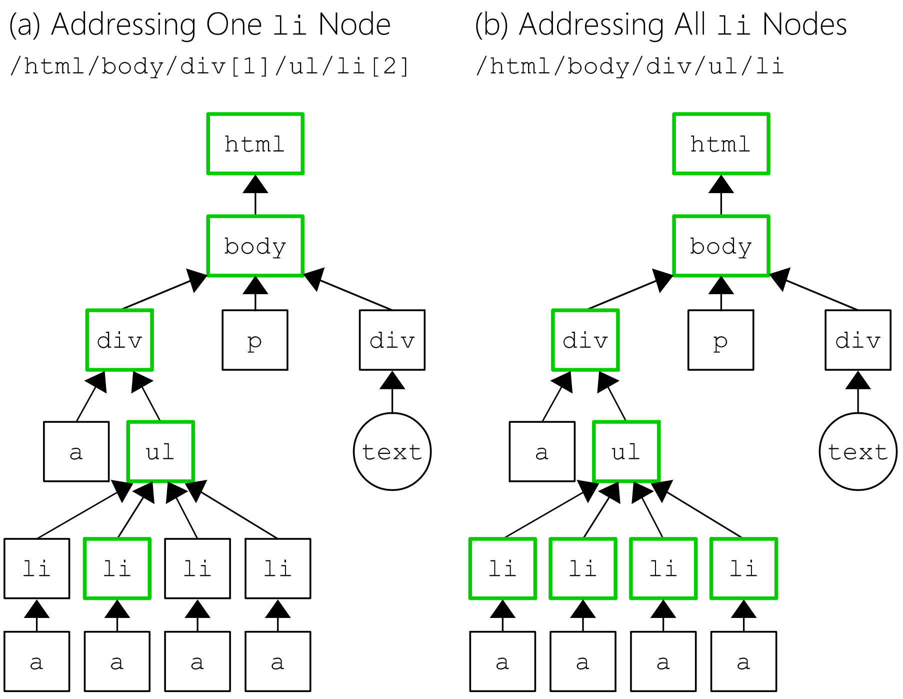


Figure 5.7: Example Web Page with an XPath Addressing Only One List Node (a) and All List Nodes (b)

After the indices are removed, all XPath instances that lead to the same stripped XPath can be counted and sorted by the number of occurrences. We assume that the XPath with the most occurrences on a Web page encodes a list, since list entries are most often encoded the same way. This process is often called searching for the highest fanout of a DOM tree. For example, the sorted list of stripped XPath for the DOM tree from Figure 5.7 would be as shown in Table 5.4.

Stripped XPath	**#Occurrences**
`HTML/BODY/DIV/UL/LI`	4
`HTML/BODY/DIV/UL/LI/A`	4
`HTML/BODY/DIV`	2
`HTML/BODY/DIV/A`	1

Table 5.4: Stripped XPaths and their Number of Occurrences on a Web Page

By now, the list detector found that the XPath `HTML/BODY/DIV/UL/LI` addresses at least as many nodes as any other XPath for that Web page and is likely to be the one used to encode a list.

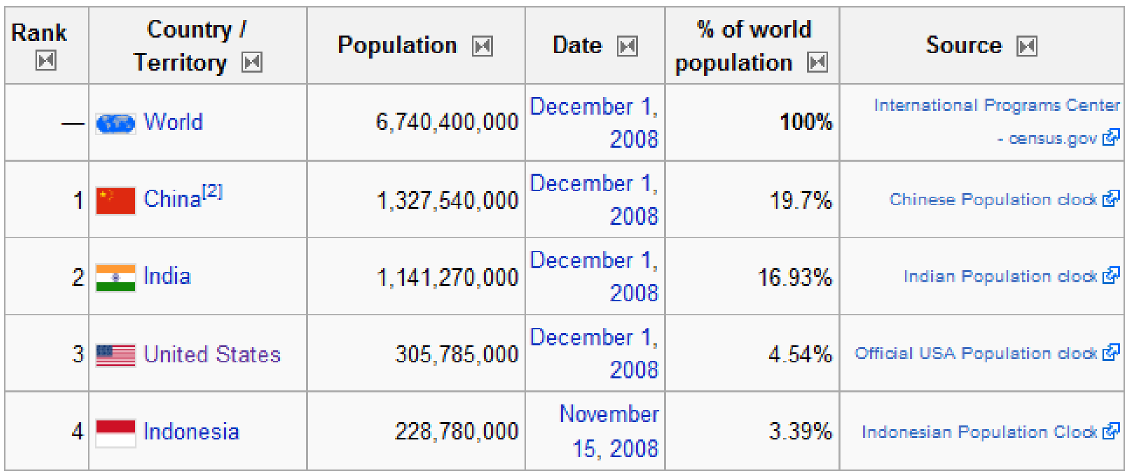
Longest XPath and Tables The list detector favors longer XPaths over shorter ones because we assume that the deeper the hierarchy, the more precise the node text and therefore less noise around an entity is extracted. Consider Table 5.4, which shows the counts of the XPaths for the example page. The second XPath has the same number of occurrences, but is longer. Thus it is favored over the first one. If the list detector finds that the entries in the longer XPath are not uniform, it takes a shorter path by removing one element after another until the top ranked XPath is reached again. If the entries for the second XPath are not uniform, the shorter but equally ranked XPath would be taken in the example from Table 5.4.

Lists are often encoded using tables, that is, using the `TABLE`, `TR`, and `TD` tags. A stripped XPath will point to every cell of a table, which is usually not helpful since often not all columns of the table are used to list entities. Figure 5.8 shows an example table[9] where the sought entities (here of the concept *Country*) are listed in column two of the table.

The list detector notices when a list is encoded in a table and tries to find the column with uniform entries. Uniformity heuristics are explained in the next paragraph. If a uniform column is found, the index is added to the `TD` tag of the stripped XPath. In Figure 5.8, for example, the XPath would be `/HTML/BODY/DIV/DIV/DIV/DIV/TABLE[2]/TBODY/TR/TD[2]/A` since only column 2 has uniform entries.

Entity Uniformity Heuristics As explained in the beginning of this section, lists must have certain features in order to be detected and used for entity extraction. Most lists are used to present something other than entities. For example, lists can contain links to blog entries, numeric values such as prices, and so forth. For this reason, heuristics have to be employed to determine whether the detected XPath really addresses a list of entities that

[9]Screenshot taken from `http://en.wikipedia.org/wiki/List_of_countries_by_population` (last accessed on 25th of March 2012).



Rank	Country / Territory	Population	Date	% of world population	Source
—	World	6,740,400,000	December 1, 2008	**100%**	International Programs Center - census.gov
1	China[2]	1,327,540,000	December 1, 2008	19.7%	Chinese Population clock
2	India	1,141,270,000	December 1, 2008	16.93%	Indian Population clock
3	United States	305,785,000	December 1, 2008	4.54%	Official USA Population clock
4	Indonesia	228,780,000	November 15, 2008	3.39%	Indonesian Population Clock

Figure 5.8: Example Table with Only One Column of Uniform Entities

should be extracted. A detected list is only used for extraction if all the following features are fulfilled:

1. Less than 15 % of the entries in the list are numeric values. Entities are usually names that have letters and sometimes numbers, but are rarely numeric values. A series of numbers is therefore not considered to be relevant for entity extraction. For example, in Figure 5.8 the table columns one, three, and five are ruled out by this heuristic.

2. Less than 50 % of the entries in the list are completely capitalized. Completely capitalized means that all characters of a word are uppercase; sometimes captions that appear between lists are completely capitalized. If too many of those entries appear, the list is not used for entity extraction.

3. On average, each entity consists of 12 words or fewer. Entity names are usually short – only one or two words. If an average of more than 12 words per list entry is found, it is very likely that the detected XPath does not point to an entity list, but rather to a set of paragraphs containing no listed entities.

4. Less than 10 % of the entries in the tables are duplicates. Lists of entities rarely contain duplicates. For example, in Figure 5.8 the table column 4 would be ruled out by this heuristic.

5. Less than 10 % of the entries in the list are missing. There are usually no gaps in lists of entities. If too many gaps appear, it is more likely that the list is an incomplete descriptive column of a table that also lists entities.

Lists in Navigation, Header, or Footer of a Web Page An entity list must appear in the content area of a Web page. Often the navigation of a Web page contains many entries that might look like a list and thus must be filtered out. Figure 5.9 depicts the problem by showing two Web pages from the same domain in (a) and (b). The green rectangles in (a) and (b) are the page specific content areas that are different for each page. The red rectangles are areas that are similar or even exactly the same since they are used for navigation, advertisement, or similar purposes.

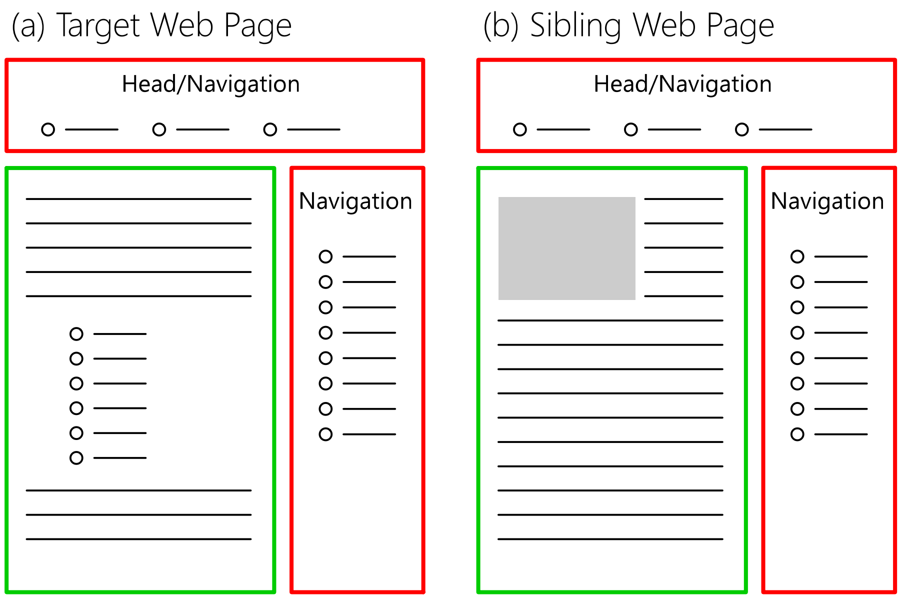


Figure 5.9: Target Web Page with an Entity List in the Green Content Area (a) and Sibling Page with Different Content in the Green Area and the Same Content in the Red Areas (b)

The list detector tries to find all elements on a Web page that do not belong to the page specific content by comparing it to a "sibling Web page". A sibling Web page is found by analyzing all links from the target Web page, that is, all contents of the `href` attribute of the `A` tag. The link to a URL with the highest similarity to the URL of the target Web page is taken as the sibling URL. The similarity between the two URLs is calculated by the number of characters from left to right that the two URLs have in common. This way the list detector is likely to retrieve a sibling Web page that has a similar structure as the target Web page.

We generate all stripped XPaths to content tags on the sibling Web page. We then compare the first 200 characters of the content targeted by the stripped XPaths to the targeted content on the sibling page with the same XPath. We use the Q-Grams[10] (Ukkonen, 1992) distance metric for string comparison. If the similarity is less than 70 %, the content is considered to be different and we assume that the XPath points to some nodes in the content area. If the similarity is greater than 70 %, the XPath is removed from the set of candidates and is not considered when searching for the highest count XPath. The similarity does not need to be 100 % because many websites use dynamic elements, such as advertisements in the navigation or header area, which yield different content for the same XPath on sibling pages. Also, it is very unlikely that the text is more than 70 % similar for pages with different content.

[10]Q-Grams can be used for approximate string matching by sliding a window of the length q over two strings. The more Q-Grams two strings share, the smaller the edit distance (higher similarity).

Pagination Detection

The pagination detector aims to find the most common paginations. It recognizes two main types of pagination, uppercase letters and a series of numbers, which are depicted in Figure 5.6. Pagination elements are almost always links, that is, using the `A` tag, since they are used to point to another page with more content. The pagination detector therefore constructs all XPaths to `A` tags and adds those to a candidate set which only consist of a digit or an uppercase letter. The process of finding the highest count XPath is similar to the one described for the list detector. The indices for the elements `A`, `TR`, `TD`, `P`, `SPAN`, and `LI` are removed since those elements are more often used to encode pagination lists. The highest ranked XPath is then taken as the XPath addressing pagination links. If a pagination consists of digits, it must have at least three entries and most of the entries must be subsequent. Therefore, a random set of linked digits is not a pagination, since page numbers are usually in an (ascending) order.

Once the pagination XPath is found, the focused crawler uses this very same XPath on every sibling page that it comes across by following the pagination URLs.

Extraction

If a list has been detected, the stripped XPath pointing to this list is used and all addressed elements are extracted as entities for the sought concept. If a pagination is found, the focused crawler uses the same stripped XPath pointing to the list of entries for every new page that is reached by following the pagination URLs.

5.2.3 List Extraction Using Seed Entities

Seed extraction aims to implicitly find pages with lists of entities by querying the search engine with seeds. Retrieval using seeds has already been used extensively by the KnowItAll system (Etzioni et al., 2005). There are some disadvantages to this retrieval mechanism since it requires correct seeds for a concept and seeds have to be combined correctly for each search query. WebKnox uses automatically obtained entities as seeds for the seed extraction technique. The seeds are either extracted with the phrase extraction technique or the focused crawl extraction technique. Thus, no human involvement is necessary.

Queries

While the focused crawl extraction technique queries a search engine by explicitly mentioning “list” or “index”, the seed extractor queries a search engine with seed entities of a concept. It is then assumed that pages with lists including the seed entities and other entities are returned from the search engine.

There are many possibilities to find “good” combinations of entities (for example, using seeds that start with the same letter), but it is most important that the seed is a correct entity. The seed extractor therefore prefers to use entities from the knowledge base that have been extracted more than once, as they are more likely to be correct extractions.

XPath Wrapper Inductor

The XPath Wrapper Inductor (XWI) aims to find the XPaths that point to the seeds and generalize it so that all entities that are encoded in the same way as the seeds are addressed and can be extracted. XWI needs at least two seeds in order to find such a "generalized XPath". The next section describes the process in greater detail.

Generalized XPath For each seed, all XPaths that point to the seed occurrences on a Web page are constructed. Each seed can potentially occur more than once and thus more than one XPath per seed may be constructed. After all XPaths for all seeds have been constructed, we search for a generalized XPath by comparing each index for each element of the XPath. Any multiple indices are deleted, which increases the number of elements addressed by the XPath. Figure 5.10 shows this process for two seeds. In (a) and (b) the XPaths to `Seed1` and `Seed2` are marked with green rectangles in the DOM tree respectively. Both XPaths are the same until the last index, indicating that more nodes with the same structure exist. The index is deleted and in Figure 5.10 (c) all siblings of the two seeds are addressed by the generalized XPath. If one or more seeds appear in different elements on the Web page, the stripped XPath with the highest count is taken.

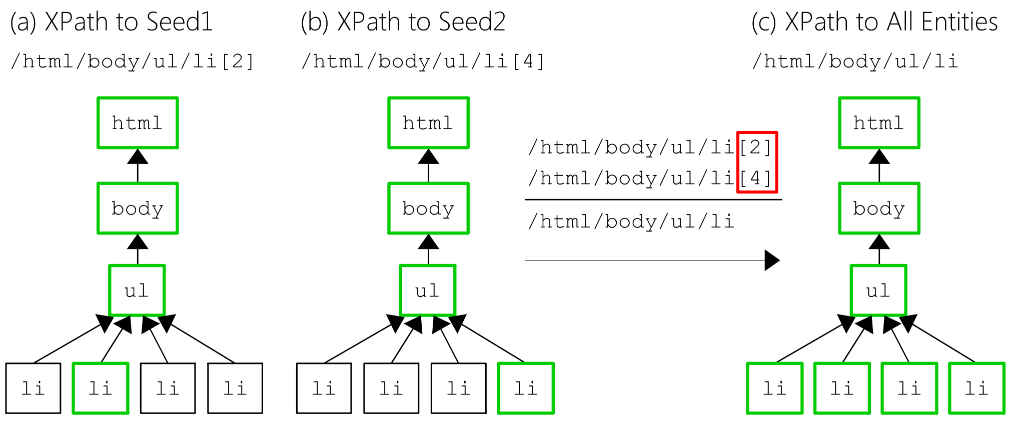


Figure 5.10: Example of Generalizing an XPath from Two Seeds (a) and (b) to Address All Target Nodes (c)

Affixes XWI occasionally uses prefixes and suffixes around the seeds. This is necessary because the generalized XPath addresses the complete tag content, even though there is sometimes information around the seed that should not be extracted. Figure 5.11 visualizes the problem. In (a) the HTML markup for a small Web page is shown. Two seeds are given: *Japan* and *New Zealand*. The generalized XPath for that example is `/HTML/BODY/UL/LI`, as shown in Figure 5.10, which addresses all `LI` tags. To avoid extracting the number before and after the seed, a two-character prefix and suffix is constructed around the seed instances, illustrated as red characters in Figure 5.11 (b). The complete wrapper for this Web page now consists of the generalized XPath and the small prefix and suffix. Applying this wrapper to the example would extract the entities *USA*, *Japan*, *Australia*, and *New Zealand* without the noise (rank and population) around the entities.

(a) Seeds in HTML

```
<html>
    <body>
        <ul>
            <li>1. USA - 301,100,000</li>
            <li>2. Japan - 127,400,000</li>
            <li>3. Australia - 21,700,000</li>
            <li>4. New Zealand - 4,100,000</li>
        </ul>
    </body>
</html>
```

(b) Prefix and Suffix for Each Seed

```
2._Japan_- 127,400,000
4._New Zealand_- 4,100,000
```

Figure 5.11: HTML Markup of a Web Page with Two Seeds Marked Green (a) and Two-character Prefix and Suffix around the Seeds (b)

Uniformity Check The extraction results of the XWI are also checked for uniformity (see Section 5.2.2) to ensure that fewer incorrect lists of entities are extracted. XWI aims for high precision and prefers to extract nothing from a Web page than a few correct entities with low precision.

Extraction

Unlike the Affix Wrapper Inductor (see Figure 5.3), XWI only works for (X)HTML documents that can be represented with a DOM tree. Since some documents are plain text, WebKnox makes use of both wrapper techniques to cover a greater variety of documents. For every (X)HTML document, WebKnox uses XWI to create a wrapper and extract entities; if the document is a text file, WebKnox utilizes the Affix Wrapper Inductor for extraction. Text files are mainly of the types TXT, DAT, or CSV.

5.2.4 Named Entity Extraction from Plain Text

Our hypothesis states that we can train an NER from the Web using only seed lists for each type as learning input. In order to test our hypothesis, we designed a named entity recognizer that can be trained with sparsely-annotated training data. Usually, supervised NERs need to be trained on completely annotated input data, such as the CoNLL 2003 dataset, and each token must be labeled by an expert. In the case of the CoNLL corpus, this means that every token has either the label `PER` (*Person*), `LOC` (*Location*), `ORG` (*Organization*), `MISC` (*Miscellaneous*), or `O` (no entity / outside). As discussed in the previous section, compiling such a training set is tremendously labor intensive. Our NER must therefore be able to learn from sparsely-annotated texts in which not every token has a label. Thus, we have an open world instead of a closed world as in supervised learning. If a token has no label, we cannot assume that it is not an entity.

Training the NER

We divide the training on sparse data into three parts: (1) creating the training data, (2) training the text classifier, and (3) analyzing the contexts around the annotations.

Creating the Training Data The input for our named entity recognizer is not a completely annotated corpus, but rather a set of "seed entities". For example, for the concept *Actor* we could use *Jim Carrey* as one of the seeds. While there are no limitations on the selection of the seeds, popular, and more importantly, unambiguous entities with many mentions on the Web are the preferred input. Figure 5.12 shows the algorithm for creating the training data. We have n concepts $C = [S_1, ..., S_n]$, each with a set of seeds S. We now query a search engine[11] with the exact seed name and its concept and take the top URLs from the result set. Each of the result URLs is downloaded, and we extract the textual content of the page using the Palladian Content Extractor (Urbansky et al., 2011a), removing the header, footer, and navigational elements to acquire only the "main content" of the Web page. Next, we annotate all our seed entities for the given concept in the text and save it. We use XML annotations in the form of `<TYPE>seed</TYPE>`. The last operation is cleaning and merging the annotated text files into one. In the cleaning process, we remove all lines that have no annotated seed, are shorter than 80 character, or have no context around the seed.

```
begin
  mentions := 50;
  for $c$ := 1 to $|C|$
    $S := C[c]$
    for $s$ := 1 to $|S|$
      seed := $S[s]$
      concept := $S_c$
      URLs := querySearchEngine(seed,concept,mentions)
      for $u$ := 1 to $|URLs|$
        webPage := download(URLs[u])
        text := extractPageText(webPage)
        for $s$ := 1 to $|S|$
          text := annotateSeed(S[s],text)
          save(text)
        end
        cleanAndMerge()
      end
    end
  end
end
```

Figure 5.12: Pseudocode for Generating Training Data

[11] We used `http://www.bing.com` (last accessed on 25th of March 2012) in our experiments.

Train the Text Classifier After generating the training data, we train a dictionary-based text classifier using the seed entities and the surrounding contexts that we found on the Web. Unlike many other NERs, we do not employ a large feature vector of numeric features such as TFxIDF, token position, or length. Instead, we treat the entity classification purely as a text classification problem. The input for our text classifier is a sequence of characters that is then divided into n-grams. The dictionary is built by counting and normalizing the co-occurrences of one n-gram and an entity type. The dictionary might then resemble Table 5.5 in which each column is an entity type (*Person*, *Location*, and *Product*) and each row is an n-gram. In each cell, we now have the learned relevance for each n-gram and entity type $relevance(ngram, entityType)$. The sum of the relevances in each row must add up to one. The n-gram "John" is more likely to indicate a *Person* ($relevance(John, Person) = 0.9$) than a *Location* ($relevance(John, Location) = 0.05$), while the n-gram "Gibson" is a little more likely to be a *Person* than a *Product* ($relevance(Gibson, Person) = 0.5$.

N-Gram	**Person**	**Location**	**Product**
John	0.9	0.05	0.05
Indiana	0.1	0.85	0.05
Gibson	0.5	0.1	0.4

Table 5.5: N-gram Dictionary with Relevances for Entity Types

We build four dictionaries of this kind. One holds only the seed names and their relevance with the entity types. A second one uses the complete context before and after the annotation within a certain window size. A third one uses only the three context words before and after the mention of the seed entity. A fourth one holds case signatures for all tokens in the training data. We store three case signatures, "A" for completely uppercase words, "Aa" for capitalized words, and "a" for lowercase words. We will show how we use these dictionaries to classify entity candidates in Section 5.2.4.

Analyze Contexts In addition to training a text classifier on the context around the seeds, we also build a dedicated context dictionary with the context patterns. The rationale behind this approach is that the sequence of words before or after a word are good indicators for the entity type. For example, consider the following phrases: "`X` traveled to `Y`", "`X` was born in `Y`", or "`X` came back from `Y`". All of these two to three word contexts indicate that Y might be a location when used as a left context pattern and that X might be a person when used as a right context pattern. We therefore build a dictionary similar to the one shown in Table 5.5, but with context phrases instead of n-grams. This approach is similar to the one described by Fleischman and Hovy (2002). We use context phrases with a length between one and three words. Also, we map all numeric expressions to the word `NUM`. This gives us a higher recall in context phrases with numbers. For instance, since it does not matter whether the phrase is "`X` paid 2 dollars" or "`X` paid 3 dollars" we capture "paid `NUM` dollars" as the context phrase. The context dictionary will be used in the last step of the entity recognition in Section 5.2.4.

Using the NER

Once we have trained our named entity recognizer on the automatically generated training data, it is ready to be used. We divide the recognition process into three parts: (1) entity detection, (2) entity classification, and (3) post processing. These steps are executed sequentially on each given text.

Entity Detection In the entity detection phase we need to find entity candidates in the text. An entity candidate is a sequence of characters of an unknown type. The output of the entity detection phase is a list of candidate annotations. For instance, given the text "John Hiatt is a great musician from Indiana, USA", we expect the entity detector to tag the candidates as "`<CANDIDATE>`John Hiatt`</CANDIDATE>` is a great musician from `<CANDIDATE>`Indiana`</CANDIDATE>`, `<CANDIDATE>`USA`</CANDIDATE>`". Our NER is intended to work for English language texts only. We therefore employ rule-based detection using regular expressions to find possible entity mentions. In particular, our expression covers sequences of capitalized words, but also allows a few lowercase words. For example, it covers "of" in order to detect *United States of America.* For each detected entity candidate, we perform the same feature generation as shown before.

Entity Classification The second phase is the classification of the candidates. To classify an entity candidate, we again create all n-grams, look up the relevance scores in the dictionary, and assign the entity type with the highest score S to the candidate. The score for each entity type and given entity candidate is calculated as shown in Equation 5.1 and 5.2 where $N_{candidate}$ is the set of n-grams for the given entity candidate and T is the set of possible types that the NER was trained to recognize.

$$S(type \mid candidate) = \sum_{n \in N_{candidate}} relevance(n, type) \tag{5.1}$$

$$entityType = \arg\max_{type \in T} S(type \mid candidate) \tag{5.2}$$

Post Processing In the last phase of entity recognition, we filter the classified entities, change their boundaries if necessary, and reclassify their types using the learned patterns. We apply the following six post processing steps:

1. We remove all entities that are date fragments, such as *Monday* and *July.* This step is necessary because these fragments are capitalized like the entities we actually want to detect, and because they fall under the tagging rules of the entity candidate detector. We lose a little recall since there are people named "April" and movies with the name "August", but the precision rises significantly more than the recall drops when applying this filter.

2. We remove those date fragments from other entity candidates and change their boundaries. For instance, "July John Hiatt" becomes the entity *John Hiatt.*

3. In the entity detection phase, we will generate many entity candidates that consist of words from the beginnings of sentences. For instance, with "This is a sentence", we would generate "This" as a candidate entity, which it is not. To filter out these incorrect detections, we employ the case dictionary that we have generated in the training phase. We calculate the ratio of uppercase to lowercase occurrences and remove the entity if the ratio is lower than or equal to one. The rationale behind this approach is that capitalized words at the beginning of sentences might appear more often in a lowercase form indicating that they are not entities. The word "This" from the example has a very low uppercase to lowercase ratio since it most often appears in the lowercase form. We borrow this idea from Millan et al. (2008).

4. We correct the classified entity type when we found the entity in the training data but classified it incorrectly.

5. We now apply the information from the context dictionary that we trained. Again, we take the context around the candidate and look up the probability of each entity type matching the context words. We merge the outcomes of the entity classification, the context classification, and the context words to reassign the most likely entity type to the candidate. For example, if we classified "Paris" as a person, but the context "born in Paris" now suggests that the candidate is much more likely a location, the context classifiers overrule the candidate-only classifier and switch the label. Next, we use our set D of dictionaries. These are (1) candidate-only, (2) context n-grams, and (3) context patterns. As shown in Equation 5.3, we combine the scores. We then use Equation 5.2 to get the most likely entity type for the candidate.

$$S(\mathit{type} \mid \mathit{candidate}) = \sum_{d \in D} S_d(\mathit{type} \mid \mathit{candidate}) \tag{5.3}$$

6. In the last step, we use the learned left context information to change the boundaries on multi-token candidates. For example, we have detected the candidate "President Obama", but knowing that the token "President" appeared many times before the actual entity in the training set, we drop this token and change the boundaries so that the candidate represents "Obama" only.

5.2.5 Entity Extraction from the Semantic Web

The previously described entity extraction algorithms all use the Visible Web (compare Section 2.2.3) as their extraction corpus. In the Visible Web, unstructured and semi-structured documents (see Section 2.2.1) dominate. The Semantic Web is growing at a fast rate and contains billions of structured data entries that can be used as a extraction corpus too. The Semantic Web Extractor (SWE) is targeted at exactly this corpus with the goal of extracting as many correct entities for our given ontology as possible. This task is often referred to as entity list completion.

Querying the Semantic Web

Information pieces on the Semantic Web adhere to ontologies and therefore make it easy for machines to read and process the data. However, the information is highly distributed over thousands of Web pages. The Semantic Web contains a few major sources, such as DBpedia and Freebase (see Figure 1.2), but Web pages containing RDFa, for example, provide more triples for the Web of Data. In the conception of the algorithm, we assume that there is an index[12] or triple store that aggregated triples from many different sources. We then use this index as the single query point, removing the need to work with distributed data.

Figure 5.13 shows the different processes in the SWE. The next sections explain these steps in more detail.

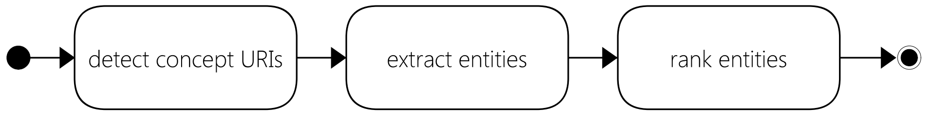


Figure 5.13: Overview of the Processes for the Semantic Web Entity Extraction

Detecting Ontology Concepts with Seed Entities

Figure 5.14 illustrates the first step of the SWE in greater detail.

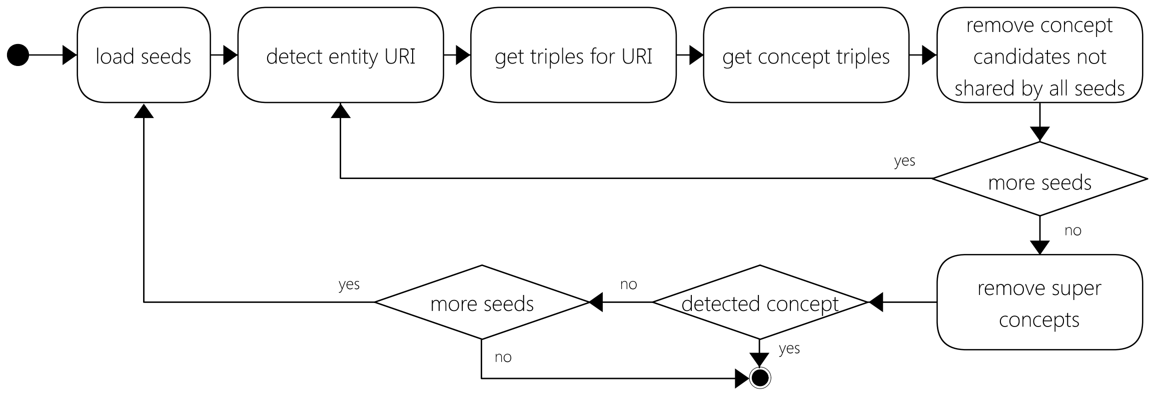


Figure 5.14: Concept URI Detection Process of the Semantic Web Entity Extractor

Having only the concept names (*concept*) and a few instances per concept ($SeedSet_{concept}$) we need to find out which ontological concepts the seeds belong to. We therefore load a combination of seeds and query the index with each *seed* and its *concept*. We then find the *seedURI* that is about our seed and extract all triple objects that might state the type ($type \in TypePredicateSet$) of the entity. Next, we put all found type candidates in a candidate list *TypeCandidateList*. After doing this for all the seed entities, we need to determine to which of the concepts in our candidate list all the seed entities belong. We therefore eliminate all concepts in the set that appeared fewer times than the number of seed entities we have.

[12]In our later experiments we will use `http://sindice.com` (last accessed on 25th of March 2012).

Furthermore, we might have extracted concepts and their super concepts. We want the most detailed concepts and need to remove their broader super concepts. We do this by resolving each concept URI and eliminating their super concepts, or eliminating the concept if we find out that it is a super concept of another concept in our candidate set. If the *TypeCandidateSet* is not empty after the process, we move on to entity extraction. If we have not detected a concept, we use a different combination of seeds and try again until we find at least one common concept. To make this process clear, we provide the following example:

Let us assume we have the following seed entities of the actor concept: $\mathit{SeedSet}_{\mathit{Actor}} = \{\mathit{Jim\ Carrey}, \mathit{Josh\ Brolin}\}$. First, we search the index for URIs with information about these seeds. From the result list, we now need to find the URI that is most likely about the seed entity. We do this by detecting the label of the subject that is described by the URI (see Section 5.2.5). If the label matches our seed entity name exactly, we take the URI as the subject and query the index for all triples that belong to that subject.

In our example we have found that the URI `http://rdf.freebase.com/ns/en.jim_carrey` matches our seed *Jim Carrey*. We would then retrieve the following triples for this URI (`s` is the subject, `p` is the predicate, and `o` is the object):

```
s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://rdf.freebase.com/ns/common.topic.image>
o: <http://rdf.freebase.com/ns/wikipedia.images.commons_id.4945051>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://rdf.freebase.com/ns/base.popstra.celebrity.supporter>
o: <http://rdf.freebase.com/ns/m.064hfhb>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://rdf.freebase.com/ns/film.actor.film>
o: <http://rdf.freebase.com/ns/m.0jykww>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/film.actor>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/award.award_nominee>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/people.person>
```

In the next step, we need to find out which type our *seed* is. We can use different vocabularies to find the type of the entity. We want the algorithm to extract from arbitrary ontologies and use only the following ontology independent predicates in the *TypePredicateSet*:

```
p: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

```
p: http://www.w3.org/2004/02/skos/core#subject
p: http://purl.org/dc/terms/subject
```

We remove all triples that do not contain a predicate which is element of the *TypePredicateSet*. We are now left with the following triples:

```
s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/film.actor>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/award.award_nominee>

s: <http://rdf.freebase.com/ns/en.jim_carrey>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/people.person>
```

After processing the first seed, the *TypeCandidateList* contains the following URIs:

```
o: http://rdf.freebase.com/ns/film.actor
o: http://rdf.freebase.com/ns/award.award_nominee
o: http://rdf.freebase.com/ns/people.person
```

We can now process our next seed, *seed* = *Josh Brolin*. We use the same procedure as with the first seed with one difference: when finding the *seedURI* for the *seed*, we limit the results to URIs from the same ontology that the first URI is from. This step is necessary for finding the common entity type among all seeds element *SeedSet*. For example, if the first *seedURI* is from DBpedia `http://dbpedia.org/resource/Jim_Carrey` and the second one is from Freebase `http://rdf.freebase.com/ns/en.josh_brolin`, their candidate types will not match since both ontologies are likely to use their own terms for the type. For example, in DBpedia *Jim Carrey* is of type `http://dbpedia.org/ontology/Actor` while in Freebase his type would be `http://rdf.freebase.com/ns/film.actor`.

After processing our second seed, *seed* = *Josh Brolin* and finding its *seedURI* `http://rdf.freebase.com/ns/en.josh_brolin`, we can add types to the *TypeCandidateList* from the following triples:

```
s: <http://rdf.freebase.com/ns/en.josh_brolin>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/film.actor>

s: <http://rdf.freebase.com/ns/en.josh_brolin>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/tv.tv_guest_role>
```

```
s: <http://rdf.freebase.com/ns/en.josh_brolin>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/people.person>
```

After adding the types to the *TypeCandidateList*, it contains the following URIs:

```
o: http://rdf.freebase.com/ns/film.actor
o: http://rdf.freebase.com/ns/award.award_nominee
o: http://rdf.freebase.com/ns/people.person
o: http://rdf.freebase.com/ns/film.actor
o: http://rdf.freebase.com/ns/tv.tv_guest_role
o: http://rdf.freebase.com/ns/people.person
```

We now want to find all types that all seeds from the *SeedSet* have in common, that is, we remove all types from the *TypeCandidateList* that occur fewer times than |*SeedSet*|. After this step our set is reduced to two types:

```
o: http://rdf.freebase.com/ns/film.actor
o: http://rdf.freebase.com/ns/people.person
```

Our goal, however, is to find the most precise concept among all seeds. In a last step, we resolve the URIs from the *TypeCandidateSet* and remove all super concepts from each candidate from the set. We remove all concepts that are objects to the predicate `http://www.w3.org/2000/01/rdf-schema#subClassOf` for a subject from the *TypeCandidateSet*. In our example, we can remove freebase:people.person from the set after discovering that freebase:film.actor is a subclass of that concept. Our final detected concept URI in this example is therefore `http://rdf.freebase.com/ns/film.actor`.

Extraction of Entities

Figure 5.15 shows the second step of the SWE in greater detail.

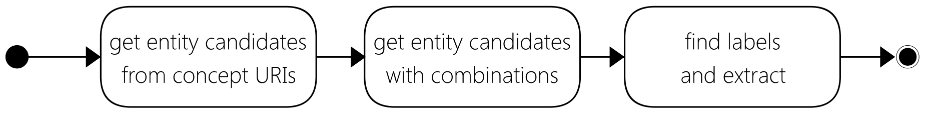

get entity candidates from concept URIs
get entity candidates with combinations
find labels and extract

Figure 5.15: Entity Extraction Process of the Semantic Web Entity Extractor

After we have detected the common type URIs for all the seeds from the *SeedSet*, we can query the index to find more entity mentions and extract them. First, we collect an *EntityCandidatesSet*. Figure 5.15 shows the first way to accomplish this. We resolve each URI from the *TypeCandidateSet* and add all subject URIs to the *EntityCandidatesSet* that have a predicate which is element of *TypePredicateSet* and have a URI that is element of the *TypeCandidateSet* as object. In a second step, we query the index for all combinations of predicates and detected types. This means we have $numQueries = |TypePredicateSet| \times |TypeCandidateSet|$. In our example, we would query the index with one detected concept URI (freebase:film.actor) and the three URIs from the *TypePredicateSet*. From each result, we add the subject URI to the *EntityCandidatesSet*.

After having collected candidate URIs in the *EntityCandidatesSet*, we need to find the corresponding label for each of the candidate URIs. To retrieve the label, we resolve the entity candidate URI and analyze all triples with a predicate element of *LabelPredicateSet*. Again, we want to be ontology independent and use only generic vocabulary[13]. The *LabelPredicateSet* contains the following predicates:

```
p: http://www.w3.org/2000/01/rdf-schema#label
p: http://www.purl.org/dc/elements/1.1/title
```

If we find a label we extract the entity. If there are several labels, we search until we find the English one. Usually the language is denoted with a `@LANGUAGE_CODE` after the literal. If we are unable to find the label by analyzing the entity candidate's triples, we try to guess the label from the entity candidate URI itself. The URI does not have to be human readable and contain the entity label but sometimes it still does. For example, the following two URIs describe the same entity *Jim Carrey*:

```
s: http://sw.opencyc.org/concept/Mx4rvo3dlZwpEbGdrcN5Y29ycA
s: http://dbpedia.org/resource/Jim_Carrey
```

If we were not able to find a proper label, we take the last part of the URI (everything after the last slash), clean the string (replace underscores with spaces), and perform a plausibility check. For this last check, we use simple heuristics. We consider a label implausible if

- the longest consecutive string in the label is longer than 25 characters, or
- the longest consecutive string in the label is longer than 15 characters and contains more than two digits, or
- the label starts with an `@` symbol.

In the example above, if we did not find a proper label for either of the two URIs, we would consider "Mx4rvo3dlZwpEbGdrcN5Y29ycA" implausible, but would extract the term "Jim Carrey" from the DBpedia URI.

We perform these steps for each candidate entity from the *EntityCandidateSet* and proceed to the ranking step once we have processed all candidates.

[13]An example of an ontology dependent vocabulary is `http://rdf.freebase.com/ns/type.object.name`, which is similar to `http://www.w3.org/2000/01/rdf-schema\#label`, but is primarily used by Freebase and not by other ontologies.

Ranking Extractions

Figure 5.16 shows the third and last step of the SWE in greater detail.

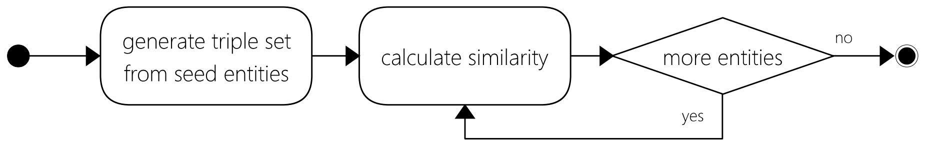


Figure 5.16: Entity Ranking Process of the Semantic Web Entity Extractor

After we extract entities from the *EntityCandidateSet*, we have an *ExtractedEntitiesSet*. To ensure we only use the most likely entities, we need to rank the entities from the set.

To rank the entities, we calculate the similarity of each entity from the *ExtractedEntitiesSet* with the entities from the *SeedSet*. The more similar the entity, the higher the rank. As a similarity function, we use the Jaccard similarity coefficient as shown in Equation 5.4, where $Triples_{seeds}$ is the union of the sets of all predicates and objects (URIs and literals) that were found for the seed entities from the *SeedSet*. $Triples_{entity}$ is the set of all predicates and objects that were found for the entity that is being ranked.

$$Rank(entity) = \frac{Triples_{seeds} \cap Triples_{entity}}{Triples_{seeds} \cup Triples_{entity}} \tag{5.4}$$

The rationale behind this similarity approach for ranking is that ranking should be relative to the entities from the *SeedSet*. For example, if we used only comedy actors as seeds, other comedy actors should be ranked higher than musical actors. The amount of predicates and objects that comedy actors have in common is usually higher than the amount of predicates that musical actors and comedy actors have in common.

5.3 Evaluation

In this section, we evaluate the performances of the extraction techniques. We compare them against each other in regards to precision and to the number of expected extractions. Our evaluation results are all based on search indices; therefore, a recall cannot be calculated. Please revisit Section 2.3 for detailed information about evaluation measures.

While some techniques might work well for a certain concept, they might be useless for all other concepts. The goal of WebKnox, however, is to work across a set of different concepts. We therefore chose to evaluate the entity extraction techniques on a set of 17 concepts as shown in Figure 5.17.

In the following sections, we evaluate the entity extraction techniques (Section 5.3.1), the performance of the Semantic Web extraction technique for the ELC task (Section 5.3.2), and provide a more detailed evaluation and comparison of the natural language-based extractor (Section 5.3.3).

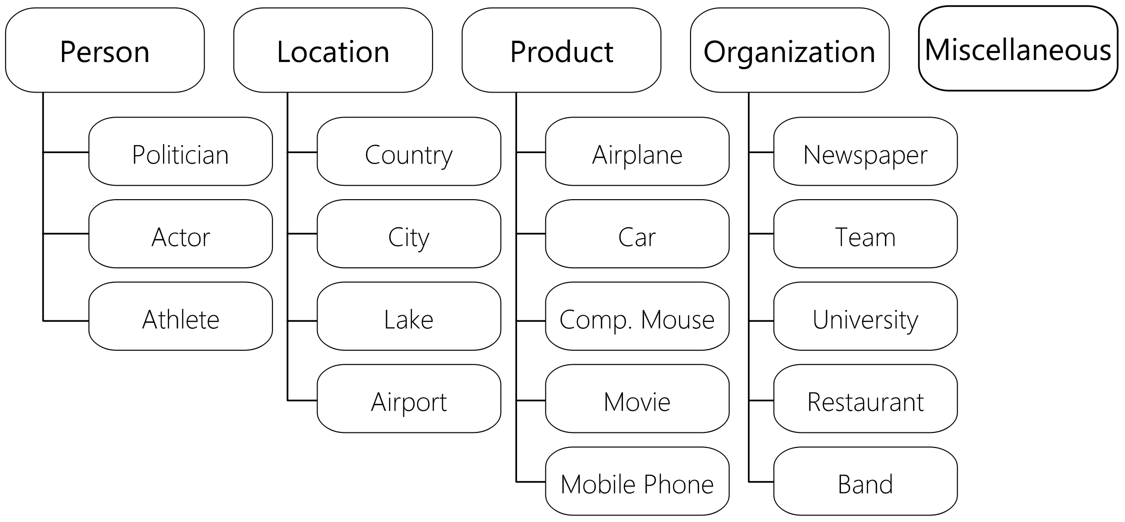


Figure 5.17: 17 Concepts for the Evaluation

5.3.1 Evaluation and Comparison of Entity Extraction Techniques

In this section, we evaluate the precision of each extraction technique across the 17 concepts from our ontology provided in Figure 5.17. Each of the five extraction techniques can be used in different settings, which influence their performance. We evaluate the precision of each technique in detail, which results in 18 different extraction technique configurations. Table 5.6 describes these configurations in detail[14] ; `CP` is the plural of a concept name and `CS` is the singular of a concept name.

Experimental Setup

To evaluate the entity extraction techniques, we applied the techniques to the Visible Web and Semantic Web in mid-August 2011.

For all extraction techniques that extract entities from the Visible Web, we used the API from Bing to perform searches and took the top 20 results per query into consideration for extraction. For the Semantic Web Extraction technique, we used Sindice as an index of the Semantic Web and used the top 10 results for extraction.

Phrase Extraction We queried with four patterns per concept and took the top 20 documents per query. This resulted in $4 \times 20 \times 17 = 1,360$ documents, which led to 3,096 extractions.

Seed Extraction We manually created a set of 20 seeds per concept (340 seeds in total) and queried with 20 random seed combinations per concept. We took the top 10 documents

[14]For the Focused Crawl Extraction we have to exclude results with “arizona” because “AZ” is the state abbreviation for the US state Arizona.

for extraction. This led to $20 \times 10 \times 17 = 3,400$ analyzed documents yielding 1,291,843 extracted entities.

Focused Crawl Extraction We queried the search engine with six queries per concept and retrieved the top 20 documents for extraction. This approach led to $20 \times 6 \times 17 = 2,040$ analyzed documents and yielded 699,616 extracted entities.

Natural Language Processing-based Extraction We used the top 20 "hot searches" from 19th of August 2011 on Google Hot Trends[15]. For each of the 20 search terms we used the named entity recognizer on the top 10 news pages delivered by Google to extract entities. In doing so, we analyzed $20 \times 10 = 200$ documents and were able to extract 4,457 entities. We call this extraction technique NLPE in the future.

Semantic Web Extraction We queried Sindice with 65 manually assigned concept URIs listed in Appendix A and a random combination of four to one seeds. First, we attempted maximally 40 combinations of four seeds. If no results were returned, we attempted maximally 40 times with three seeds. We continued until a result was returned or until we had no more concept URIs or seeds to query with. Using this approach, we posed 1,512 queries to Sindice resulting in 31,409 extractions.

Altogether, we extracted over two million possible entities across the 17 evaluation concepts. Since this amount of entities is too large for evaluation purposes, we randomly selected about 380 entities per concept (6,489 in total) to manually evaluate whether extractions were correct. This number of 380 entities per concept allows us to calculate precision scores within small confidence intervals (the exact confidence interval will be reported in the pertinent sections). We use a confidence level of 95 % to determine the confidence levels for each reported precision.

Extraction Techniques Performance by Concept

Figure 5.18 depicts the extraction precision for each of the 17 concepts and each of the 18 extraction technique configurations in form of a heat map. White signifies low precision and dark blue signifies high precision. In this particular chart, the confidence interval is about 13 % on average for a confidence level of 95 %.

The figure shows which extraction technique configuration works best for each concept. While the "2 Seeds" configuration of the Seed Extraction technique performs rather well across all concepts, the "Browse" configuration of the Focused Crawl extraction technique works particularly well for the concepts *Politician*, *Country*, *Band*, *Airport*, *Actor*, and *Movie*, but fails to extract the other concepts precisely. As long as we have one dark blue square in each row, we can extract entities from that concept with at least one extraction technique confidently. Furthermore, the figure justifies extraction technique configurations which rarely extract entities precisely but excel in at least one concept. For example, the "AZ" configuration of the Focused Crawl extraction technique generally does not extract with high precision, but is

[15]`http://www.google.com/trends/hottrends?date=2011-8-19&sa=X`, last accessed on 25th of March 2012

Technique	**Configuration**	**Explanation**
Phrase Extraction	Such As	Web pages were retrieved with the query type "`CP` such as".
	Like	Web pages were retrieved with the query type "`CP` like".
	Including	Web pages were retrieved with the query type "`CP` including".
	Especially	Web pages were retrieved with the query type "`CP` especially".
Seed Extraction	2 seeds	Two seeds were used when querying a search engine.
	3 seeds	Three seeds were used when querying a search engine.
	4 seeds	Four seeds were used when querying a search engine.
	5 seeds	Five seeds were used when querying a search engine.
Focused Crawl	List Of	Web pages were retrieved with the query type "list of `CP`".
	List	Web pages were retrieved with the query type "`CS` list".
	Index Of	Web pages were retrieved with the following query type "index of `CP`".
	Index	Web pages were retrieved with the query type "`CS` index".
	Browse	Web pages were retrieved with the following query type "browse `CP`".
	AZ	Web pages were retrieved with the two query types "`CP` 'a-z' -arizona" and "`CP` 'a to z'".
NLPE	Sparse Model	The NLPE used a sparsely-trained model with 50 seeds per concept to discover new entities.
	Complete Model	The NLPE was trained on the TUDCS4 corpus.
SWE	Assigned Classes	For each of the 17 concepts we manually assigned a set of matching RDF classes found on the Semantic Web. See Table A.1 in Appendix A for a complete list.
	Detected Classes	We used between one and four seeds to automatically detect the mutual RDF type of the seeds.

Table 5.6: Configurations of the Five Entity Extraction Techniques

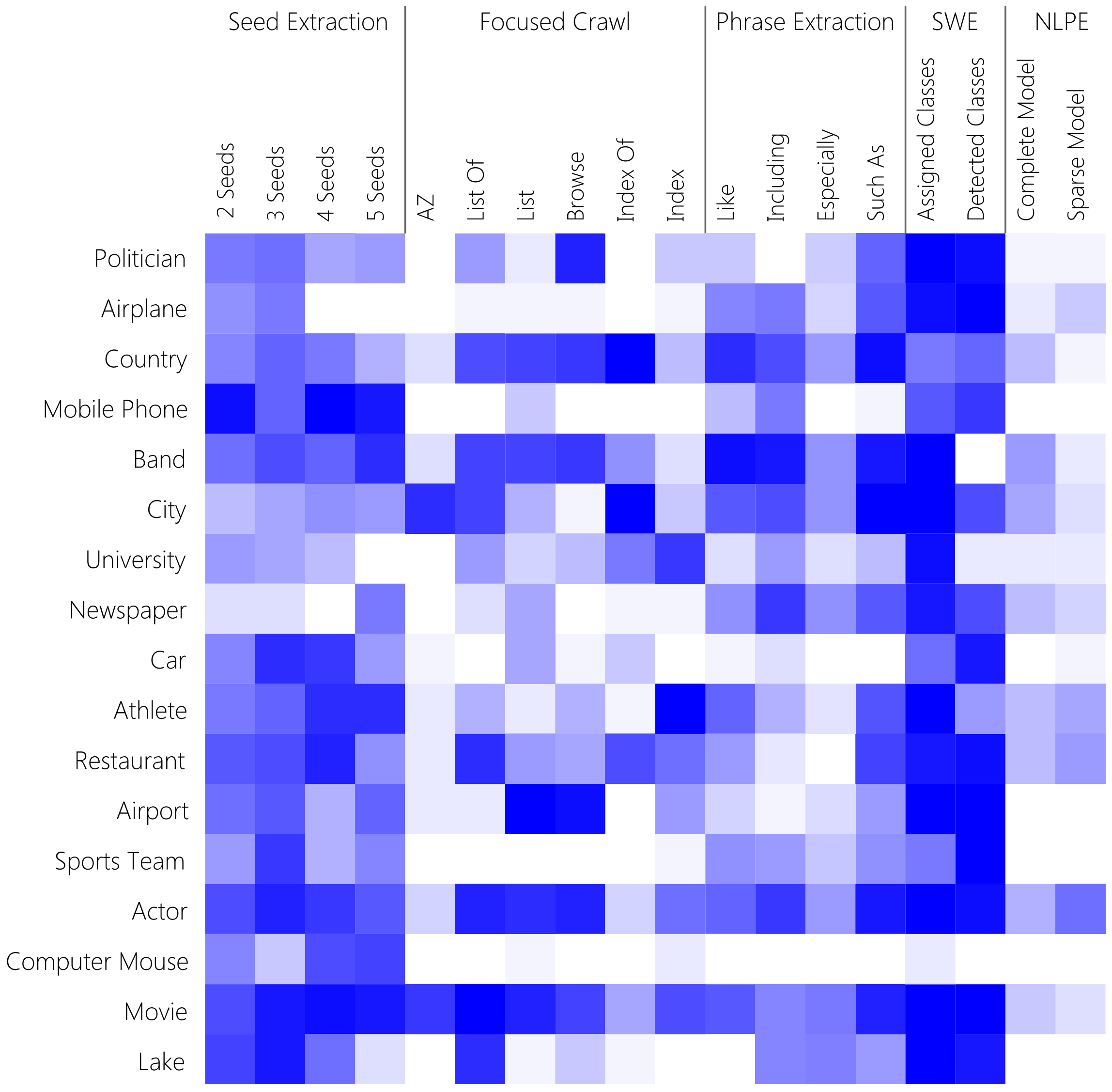


Figure 5.18: Precision in Each of the 17 Concepts by Each of the 18 Extraction Technique Configurations

still valuable because it is one of only four configurations that extracts city names with high precision.

Extraction Techniques Averaged over 17 Concepts

Figure 5.19 depicts the precision scores of each individual extraction technique configuration averaged over all 17 concepts. Additionally, Figure 5.20 shows the estimated number of correct extractions (see Equation 2.9 with $\beta = 1$).

The Semantic Web Extractor extracts entities with the highest precision of 81 % on average.

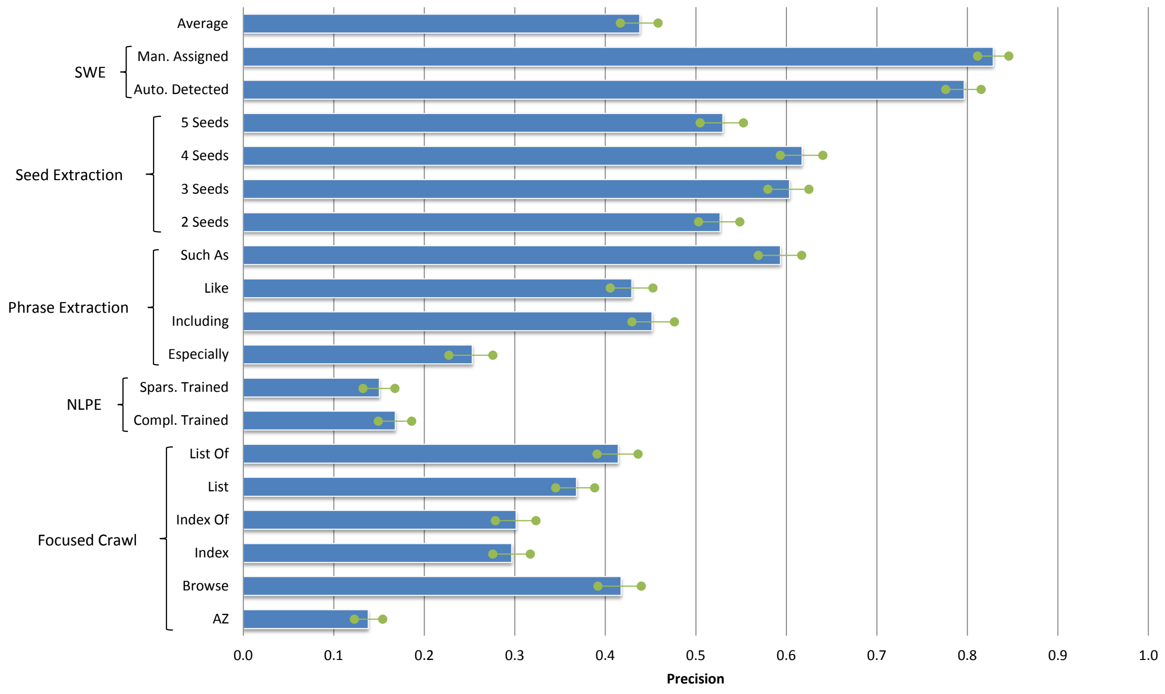


Figure 5.19: Precision of the 18 Extraction Technique Configurations Averaged over All 17 Concepts

Interestingly, the configuration with automatically detected RDF classes is only about 3 % less precise than the configuration where we have manually assigned the correct concepts. The manual effort for finding URIs does not yield much better results and could be avoided altogether.

The Seed Extraction technique is the second most precise with about 57 % precision on average. We expected the precision to increase as we used more seeds. This expectation was confirmed by the precision scores for two to four seed queries, but not when we used five seeds. In the case of five seed queries the precision dropped back to the level of two seed queries. Intuitively, the more seeds we use to search for Web pages, the more "hidden" pages will be retrieved from the search engine index. These pages might be lower in quality (therefore also more "hidden") and the extraction techniques might extract more false positives. Using three to four seeds also yields the highest number of estimated correct extractions, around 275,000.

The Phrase Extraction technique varies considerably in precision depending on which terms are used for retrieval and extraction. While more than every second extracted entity is correct (59 %) using the phrase "such as", only about every fourth extraction (25 %) is correct when using the phrase "especially". A variety of patterns is necessary, however, since the number of instances found per pattern is relatively low compared to other techniques. The number of estimated correct extractions is below 350.

The NLPE (extraction of entities from plain text) is only about 16 % precise, which we expected due to the lack of "guidance" through HTML structures or hand-crafted patterns. It

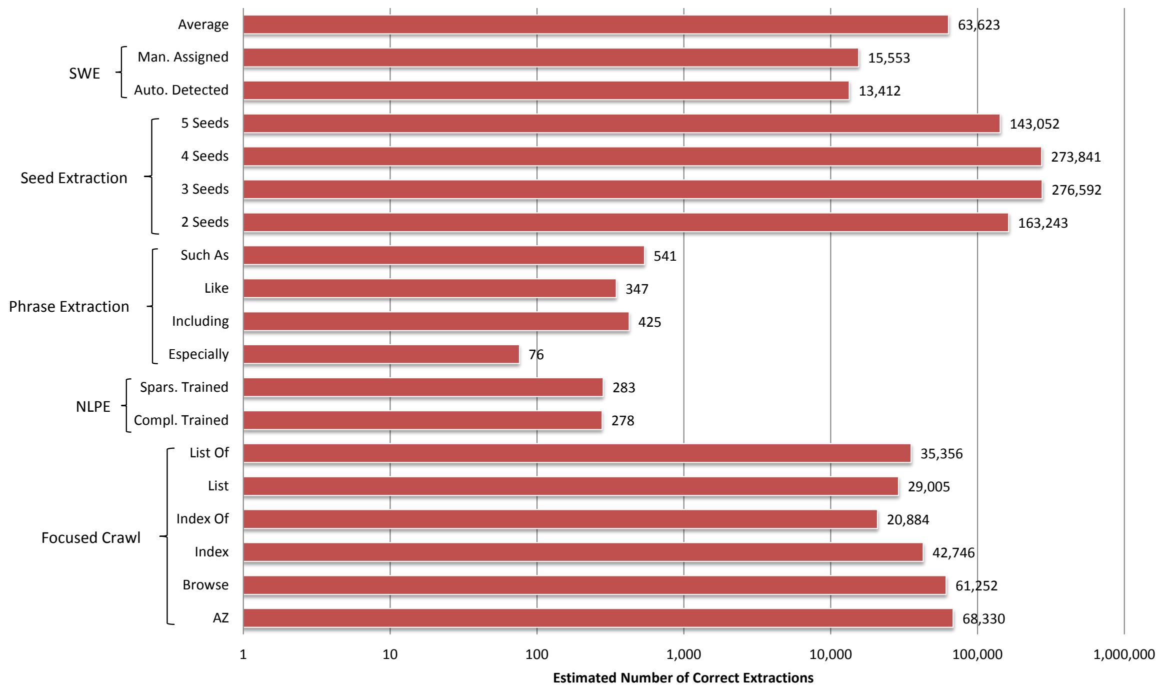
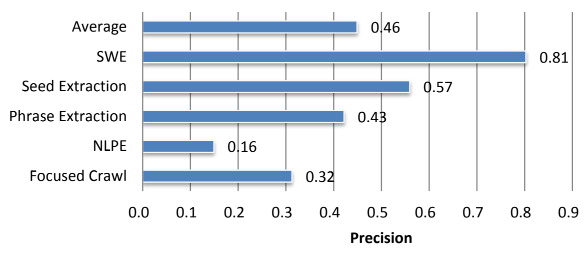


Figure 5.20: Estimated Number of Correct Extractions of the 18 Extraction Technique Configurations Averaged over All 17 Concepts

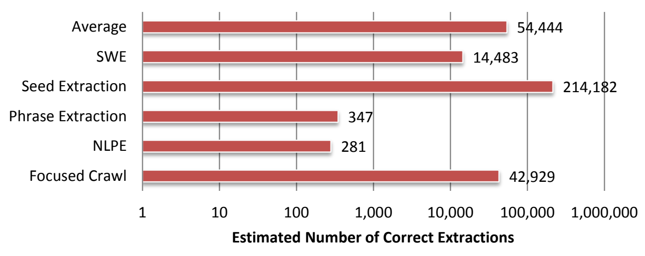
is intriguing, however, that the configuration with the model that was automatically learned on sparse data is only about 2 % less precise than the model trained on the completely annotated TUDCS4 corpus. Just as with the Phrase Extraction, the estimated number of correct extractions is comparatively low with about 300 extractions.

The extraction precision of the Focused Crawl extraction technique depends on the phrases used to retrieve the list pages. Even the difference between the retrieval phrases “list” and “list of” yielded a precision difference of almost 5 %, although this was not the case for the similar pair “index of” and “index”. The pattern “AZ” results in the lowest precision. In general, the Focused Crawl technique always tries to detect lists on retrieved pages. Query phrases that lead a search engine to bring up predominantly pages without lists will therefore decrease the extraction precision tremendously. Depending on the configuration, between 20,000 and 70,000 estimated correct instances were found.

Figure 5.21 shows the precision and the estimated number of correctly extracted instances of the five main extraction techniques averaged over their configurations and all 17 concepts. All calculated scores are within a confidence interval of about 5 % at a confidence level of 95 %.



(a) Averaged Precision



(b) Sum of Estimated Correct Extractions

Figure 5.21: Performance of the Five Extraction Techniques Averaged over All 17 Concepts

Concepts Averaged over Five Extraction Techniques

Figure 5.22 depicts how many entities were extracted across the 17 concepts and how precisely they were extracted. The data is averaged over the 18 extraction technique configurations.

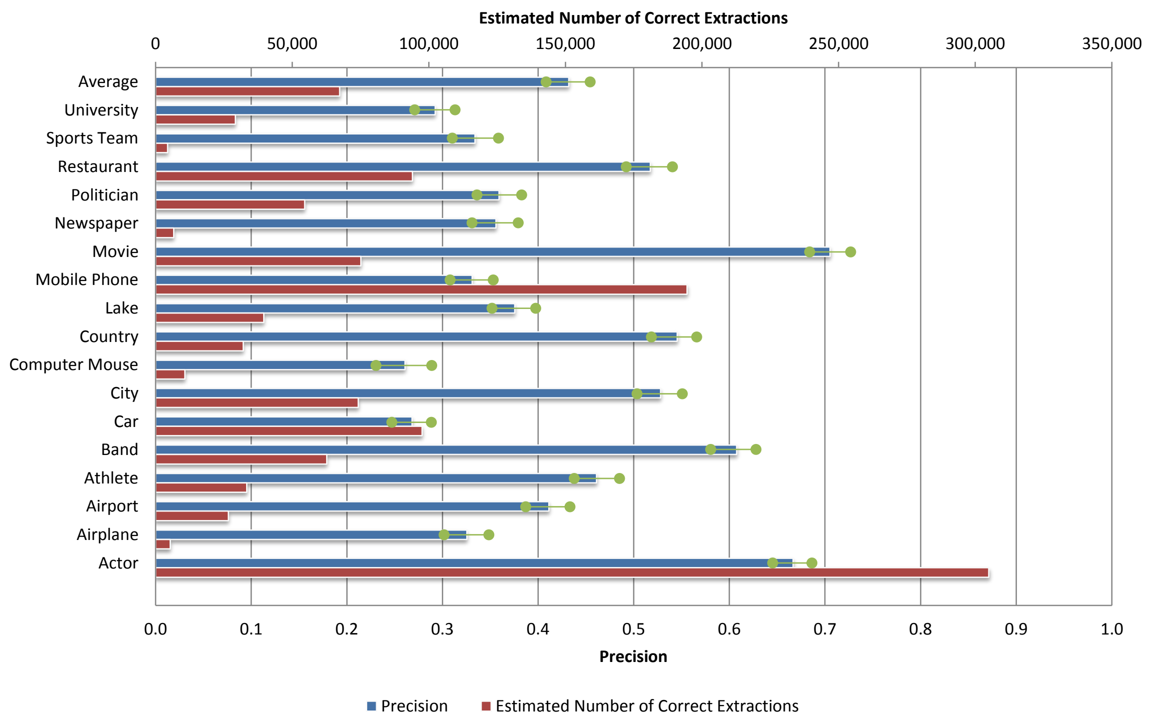


Figure 5.22: Precision in Each of the 17 Concepts Averaged over All Five Extraction Techniques

On average, about 43 % of all extractions were correct, and an estimated 67,000 correct instances per concept could be extracted[16]. We can see that extraction precision varies greatly depending on the concept. Movie names were extracted with a precision of almost 71 % while only about every fourth (26 %) computer mouse extraction was correct. An explanation for

[16]The extractions are not unique, the same entity name might have been extracted multiple times from different sources.

this gap in precision is the popularity of the entities of the concepts. Popular concepts such as movies are much more frequently mentioned on the Web than computer mice, for example. To verify this claim, compare Table 5.7 and Table 5.8.

In Table 5.7, we retrieved the hit counts on Google for the top 10 movies on the Internet Movie Database (IMDb)[17]. The average hit count is about 18.7 million indexed pages on Google.

Entity	**Google Hit Counts**
The Shawshank Redemption	5,270,000
The Godfather	30,600,000
The Godfather: Part II	1,700,000
The Good, the Bad and the Ugly	22,200,000
Pulp Fiction	11,300,000
12 Angry Men	2,060,000
Schindler's List	4,320,000
One Flew Over the Cuckoo's Nest	3,440,000
The Dark Knight	82,400,000
Inception	24,200,000
Average	18,749,000

Table 5.7: Hit Counts on Google for the Top 10 Movies on IMDb

In Table 5.8, we retrieved the hit counts on Google for the top 10 best selling computer mice on Amazon.com[18]. The average number of indexed pages containing the names of the mice is only about 1.5 million.

There are more than ten times more indexed pages for movies than computer mice. Therefore, it is also more likely that we extract a popular movie several times from different pages and get a higher precision.

Furthermore, movies, actors, and bands are often listed in databases which can be browsed on the Web. The Web pages containing lists of entities of these concepts are usually not hand-written HTML but templates with repeating patterns, making it easier to extract from them.

Again, all calculated scores are within a confidence interval of about 5 % at a confidence level of 95 %.

An average precision of 43 % is not satisfactory. We will show how to assess extracted entities in order to increase precision in Chapter 6.

[17]`http://www.imdb.com/chart/top`, top 10 movies on 5th of September 2011

[18]`http://www.amazon.com/Best-Sellers-Electronics-Computer-Mice/zgbs/electronics/11036491/ref=zg_bs_nav_e_4_172493`, best-selling computer mice on 5th of September 2011

Entity	**Google Hit Counts**
Logitech Wireless Mouse M305	2,000,000
Microsoft Arc Touch Mouse	1,660,000
Apple Magic Mouse	6,950,000
Logitech B100 Optical USB Mouse	432,000
Logitech M305 Wireless Mouse	1,310,000
Logitech Wireless Performance Mouse MX	522,000
HP 2.4 GHz Wireless Optical Mobile Mouse	763,000
Logitech Wireless Anywhere Mouse MX	405,000
Logitech Wireless Mouse M510	971,000
Logitech Wireless Trackball M570	558,000
Average	1,557,100

Table 5.8: Hit Counts on Google for the Amazon Top 10 Selling Computer Mice

Overlap of the Extraction Techniques

In previous sections, we have compared the extraction precision and the estimated number of correct extractions across the five techniques. We saw that some techniques, such as NLPE, perform rather poorly with only about 15 % precision and only about 280 estimated correct extractions. In this section, we calculate how often the extraction techniques overlap to determine whether it is even necessary to use all the techniques. Equation 5.5 shows how we calculate the overlap using Jaccard.

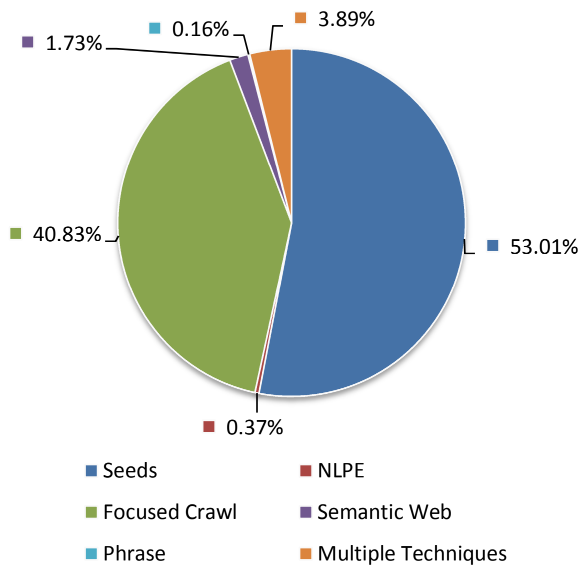
$$Overlap(T1, T2) = \frac{|T1 \cap T2|}{|T1 \cup T2|} \tag{5.5}$$

Table 5.9 lists the overlap scores between all techniques. Since the overlap is symmetric ($Overlap(T1, T2) = Overlap(T2, T1)$), only half the matrix is filled. We can clearly see that none of the techniques overlap very much.

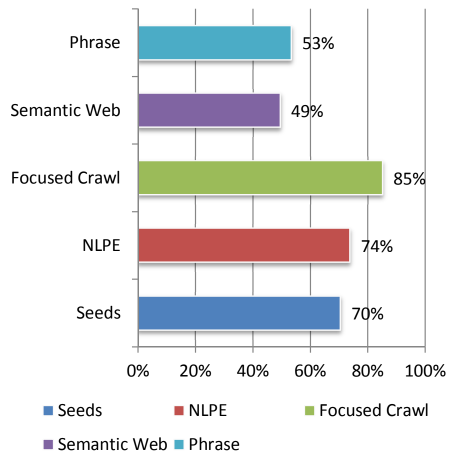
	Focused Crawl	**NLP**	**Phrase**	**Seed**	**Semantic Web**
Focused Crawl	1	0.0012	0.0021	**0.0312**	0.0130
NLPE		1	0.0101	0.0012	0.0026
Phrase			1	0.0017	0.0072
Seed				1	0.0138
Semantic Web					1

Table 5.9: Overlap Matrix of Five Extraction Techniques

In Figure 5.23(a), we visualize what percentage of the total number of about 876,000 entity extractions was extracted by only one of the five techniques[19]. For simplicity we call such an extraction a "one technique extraction". Interestingly, less than 4 % of the extracted entities were found using two or more extraction techniques; the rest of the extractions were one technique extractions. In Figure 5.23(b), we show the one technique extractions for each technique relative to the total number of extractions by that technique. Here we can see a much more balanced distribution. For example, while only 0.37 % of all extracted entities were only found by NLPE, we can also say that about three out of four extractions that were performed by NLPE are only found using NLPE. Even 49 % of the extracted entities using the Semantic Web Extraction technique were not found using a different technique. This result is interesting because a large amount of information on the Semantic Web is identical to information found on the Visible Web.



(a) Distribution of One Technique Extractions in the Entire Dataset



(b) Distribution of One Technique Extractions in Relation to Total Number of Extractions per Technique

Figure 5.23: Overlap Analysis of the Five Entity Extraction Techniques

We can conclude that even though some techniques are less precise than others, each technique is able to extract entities that cannot be found using any of the other techniques. Hence, all techniques should be used in the final system.

Overlap with DBpedia and Freebase

Thesis 1 (compare Section 1.4) of this work states that using multiple extraction techniques on arbitrary Web pages will find entities that are not already available in DBpedia. To test this hypothesis, we checked the overlap of our evaluation set of 2,885 correctly extracted entities with DBpedia entities. We found that only 1,504 of these entities are also available in DBpedia. This means that about 48 % of our extractions are not part of DBpedia.

[19]Since we do not know which entities are correctly extracted, we took all extractions into account.

We also tested how many entities overlapped with Freebase and found that Freebase did not contain 14.2 % of the 2,885 correct test entities in its knowledge base[20].

We can therefore contribute to the Semantic Web, as well as to systems that rely on large knowledge bases of entities, such as named entity recognizers.

So far, we were only able to compare the entity extraction techniques to each other. In the following two sections, we will evaluate SWE and NLPE on other research tasks. This gives us the opportunity to compare the approaches to other techniques on the same data and we can test the robustness of our algorithms when switching to different problems. We chose SWE and NLPE for this analysis, because they are the main contribution of this chapter and there are comparable approaches on other research tasks to compare against.

5.3.2 Semantic Web Extractor Experiments

We now evaluate the entity extraction approach from the Semantic Web in two further experiments on the entity list completion task. First, we use the extraction technique to complete lists of entities from our ontology of 17 concepts, and second, we compare the performance to the best-known entity list completion algorithms in literature on a smaller set of five concepts.

Entity List Completion on 17 concepts

Figure 5.24 shows the precision of Semantic Web Entity List Completion (SWELC) on a set of 17 different concepts. For each concept, we evaluated SWELC with and without the automatic concept detection step. We also manually assigned over three URIs on average from different ontologies to each concept.

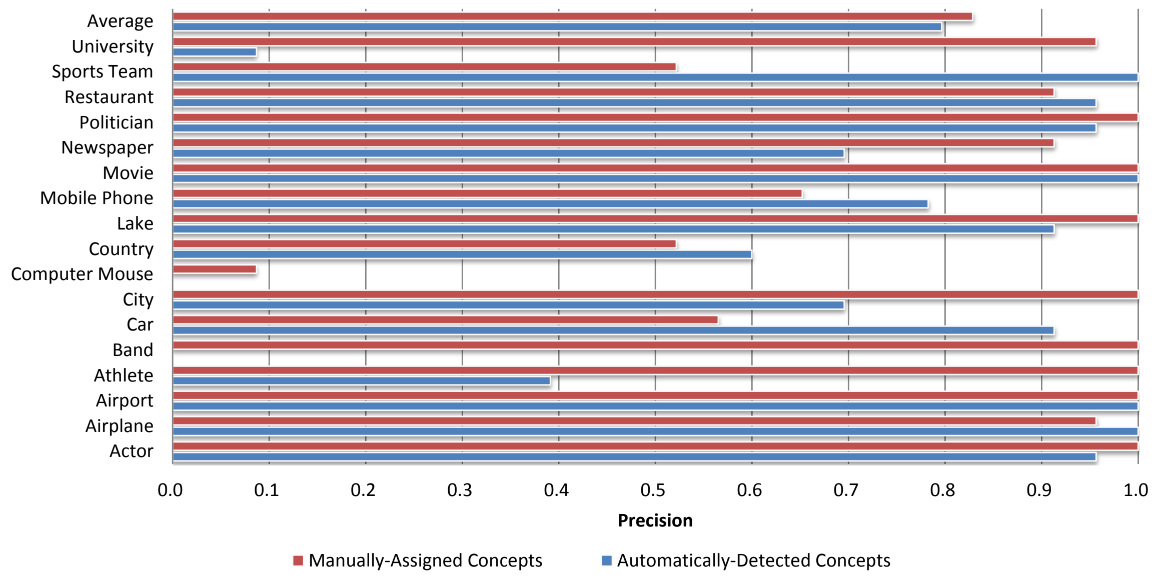


Figure 5.24: Entity List Completion Precision across 17 Concepts

[20]Date of the experiment was the 23rd of March 2012.

For the automatic detection step, we used four randomly selected seeds and tried to detect a concept with at most 40 attempts. If nothing was found, we reduced the seed set size to three, attempted 40 times again, and so on. Table 5.10 shows the number of attempts and seeds that were needed to either find a matching result or to cancel the process shown in Figure 5.15. The table also reflects the availability and absence of certain information on the Semantic Web. Products such as computer mice and mobile phones are not yet on the Semantic Web or are difficult to find. Computer mice were not even found after 320 attempts in our experiment; mobile phones were only detected after 282 attempts with only one seed, which usually yields imprecise extractions. Countries and bands, on the other hand, could be detected after just one try using four random seed entities. Ontologies such as MusicBrainz[21] for music information and the CIA world factbook[22] for geographic information provide easy access to these kind of entities.

Concept	**Attempts**	**Number of Seeds**
Actor	9	4
Airplane	151	3
Airport	194	2
Athlete	44	4
Band	**1**	4
Car	42	4
City	7	4
Computer Mouse	320	no success
Country	**1**	4
Restaurant	47	4
Lake	195	2
Mobile Phone	282	1
Movie	3	4
Newspaper	**1**	4
Sports Team	46	4
Politician	164	2
University	5	4

Table 5.10: Number of Attempts and Seeds to Successfully Detect Entities Using Semantic Web Entity Extraction

Surprisingly, the average precision of SWELC was only about 3 % lower when automatically detecting the concept compared to using the manually assigned concept URIs as shown in

[21]`http://musicbrainz.org/`, last accessed on 5th of March 2012

[22]Copied to the Semantic Web by the FU-Berlin `http://www4.wiwiss.fu-berlin.de/factbook/`, last accessed on 5th of March 2012

Figure 5.24. In fact, for several concepts, we seemed not to have chosen the optimal concept URIs manually, and the automatic detection actually found better matching URIs (e.g. for Sports Team).

Comparison to Related Work

In this section, we compare SWELC to the two state-of-the-art systems Boo!Wa! and Google Sets. We selected five concepts from different domains – *Lake* (*Location*), *Movie* (*Product*), *Politician* (*Person*), *Newspaper* (*Organization*), and *Airport* (*Construction*). Since the ELC task is mainly user oriented, we use the precision@k measure for evaluation, which correlates well with the user's satisfaction. We use $k = 10$ mainly because Google Sets seldom returns more results. For each concept, we evaluated the result lists three times, each time with three randomly selected seed entities. If a system did not produce an answer, we removed an entity and attempted again. We also removed the seed entities from the result list since they would be of no help to a human user. Table 5.11 shows the comparison of the precision@10 values for the three systems across the five concepts. Interestingly, the Boo!Wa! system was unable to answer any of the 15 queries with a result list. Google Sets performs well on the popular concept *Movie*, but is often concept dependent. Overall, SWELC performs 5 % better than Google Sets in our evaluation. It is worth mentioning that SWELC returned many more results (up to 6,000 movies, for example) than Google Sets. The problem with SWELC is the concept detection. If the right seeds were selected and the concept was detected correctly, almost all the answers were correct. If a wrong concept was detected, almost all results were wrong. Improving the concept detection will therefore yield much higher precision scores.

Concept	Boo!Wa!	Google Sets	SWELC
Lake	-	0.07	**0.50**
Movie	-	**1.00**	0.33
Politician	-	0.27	**0.33**
Newspaper	-	0.87	**1.00**
Airport	-	0.70	**1.00**
Overall	-	0.58	**0.63**

Table 5.11: Precision@10 Scores across Three Different ELC Systems for Five Concepts

5.3.3 Natural Language Processing-based Extraction Experiments

In this section, we evaluate the NLPE technique in more detail and compare it to related work. We have seen that NLPE yields rather low extraction precision in our previous experiments. We now evaluate this technique on two datasets to put the performance of our approach in relation to state-of-the-art NER techniques.

Datasets

We use two datasets for our evaluation, the CoNLL 2003 and the TUD 2011 dataset. In this section, we will describe the properties, similarities, and differences between the datasets.

CoNLL 2003 The CoNLL 2003 dataset[23] was developed for the shared task at CoNLL in 2003. The task participants were asked to provide a language-independent NER tool that can process English and German texts recognizing the entity types *Person*, *Organization*, *Location*, and *Miscellaneous*. In our evaluation, we will only use the English part of the dataset. The dataset also comes with additional data, such as list of names, POS-tags, and non-annotated data, which we also will not use in our experiments. The data is separated into training (68 %), validation (for tuning the systems, 17 %), and test (15 %). A Reuters corpus of news wire articles was annotated by the University of Antwerp for this dataset.

Table 5.12 shows the number of tagged entities per type in the three parts of the dataset. We can see that the main types are almost evenly distributed in the dataset. Altogether, almost 35,000 mentions were annotated.

Entity Type	**Training**	**Validation**	**Test**	**Total**
Person	6,560	1,832	1,602	9,994
Organization	6,276	1,341	1,642	9,259
Location	7,119	1,830	1,661	10,610
Misc	3,371	910	691	4,972

Table 5.12: Number of Tagged Entities per Type in the CoNLL 2003 Dataset

Although the CoNLL 2003 dataset is a valuable resource, it comes with a number of drawbacks. Most importantly, only three entity types in addition to *MISC* type are tagged. We have noticed that the *MISC* type very often represents a nationality identifier such as *Spanish*. Since there are many more entities that could fall into this category (*Products*, *Buildings*, *Landmarks*, etc.) it is questionable whether the annotated articles cover enough variety. This categorization is too broad for many real world applications. Therefore, the performance levels that NERs can reach on this dataset might be misleading. It is unclear whether they would work with a much finer-grained classification. Furthermore, we found that some documents in the dataset do not seem to resemble a real text, but rather look as if they were scraped table contents. Long sequences of space-separated digits or lines of completely uppercase characters seem unusual for real news articles. Lastly, we found several inconsistencies in the annotation. For example, “Dublin-born” is not annotated while “Suriname-born” is tagged as `MISC`, and sometimes even the languages German and English are mixed up.

TUD 2011 The TUD 2011 dataset[24] was developed at the Dresden University of Technology to evaluate named entity recognition on a finer-grained level. It was important to us that

[23]`http://www.cnts.ua.ac.be/CoNLL2003/ner`, last accessed on 25th of March 2012

[24]`http://areca.co/7/Web-Named-Entity-Recognition-TUDCS4`, last accessed on 25th of March 2012

we did not get data solely from one source as the writing tends to be too homogeneous. We therefore used different Web pages as sources for the dataset. Two annotators spent about 100 hours annotating 22 entity types in the dataset. In the first step, 20 seed entities per entity type were collected from Web sources. We then used the algorithm from Figure 5.12 to automatically find possible mentions of the seed entities on the Web. Documents for each entity type were then hand tagged for about four hours. We tried to assign the most specific type to each entity; if there was no fitting type we went up in the hierarchy. If nothing matched there, we had to assign *MISC*. For example, *Jim Carrey* would be annotated as *Actor* while *Bill Gates* would be a *Person* since there is no specific matching type. The 127 documents are divided equally into training and test sets.

The TUD 2011 dataset contains 22 entity types in two hierarchy levels as shown in Figure 5.17.

Table 5.13 shows the number of tagged entities per type in the training and test part of the dataset. While the five top level types are almost evenly distributed, the 17 more specific types show more variation in the number of annotated instances. This variation is due to the fact that country names, for example, also occur in documents about airplanes, but not the other way around. Altogether, 3,400 mentions were annotated.

Entity Type	**Training**	**Test**	**Total**	**Entity Type**	**Training**	**Test**	**Total**
Person	216	166	382	Car	19	14	33
Politician	54	66	120	Comp. Mouse	18	21	39
Actor	59	28	87	Movie	28	30	58
Athlete	60	78	138	Mobile Phone	24	24	48
Location	178	145	323	Organization	201	231	432
Country	143	119	262	Newspaper	70	39	109
City	142	142	284	Team	60	75	135
Lake	28	14	42	University	14	17	31
Airport	40	30	70	Restaurant	19	20	39
Product	64	44	108	Band	37	20	57
Airplane	92	118	210	Misc	243	150	393

Table 5.13: Number of Tagged Entities per Type in the TUD 2011 Dataset

Comparisons

In this section, we evaluate our NER (see Section 5.2.4) on the two datasets and compare it to state-of-the-art NERs. In particular, we compare it to LingPipe[25] (Alias-i, 2011), which employs a character language model on top of a hidden Markov model, OpenNLP[26], which

[25]`http://alias-i.com/lingpipe/`, last accessed on 25th of March 2012
[26]`http://incubator.apache.org/opennlp/`, last accessed on 18th of June 2012

follows a maximum entropy approach for tagging, and the Illinois LBJ Tagger[27] (Ratinov and Roth, 2009), which is based on conditional random fields. All these recognizers work in a supervised manner, needing manually-labeled training data. The WebKnox NER works in supervised mode (called "WebKnox" in the charts), but can also learn from sparsely-annotated training data on the Web (called "WebKnox Web" in the charts). We use the MUC evaluation scores for precision, recall, and the F1 value. Furthermore, it is important to note that evaluation is done on "unseen" data, that is, we ignored annotations in the test set which could have been learned in the training data. We do this to penalize dictionary-based recognition approaches since our goal is to extract new entities from text, not just to recognize what we know already.

Evaluation on CoNLL 2003

Figure 5.25 shows the comparison of the four NERs on the CoNLL data. All NERs were trained on 68 % training data and tested on 15 % test data from the dataset. This graphic shows what we can reach when using supervised learning with the training data.

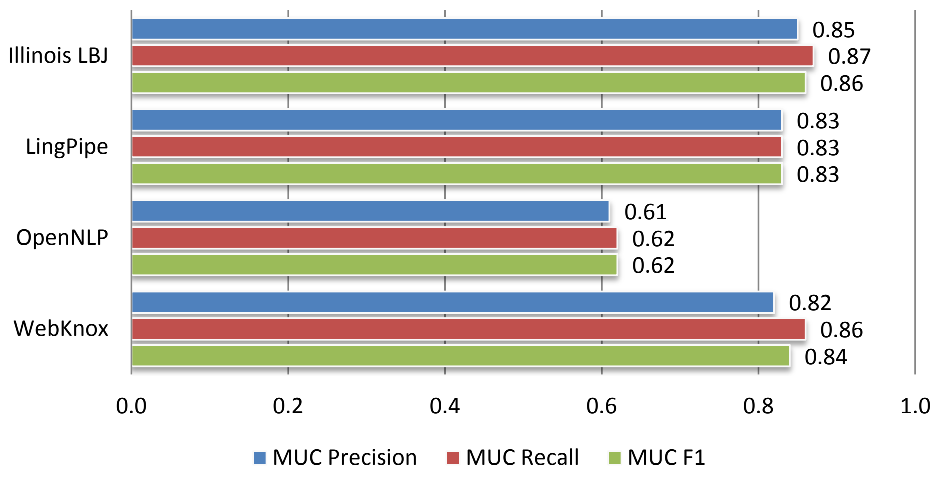


Figure 5.25: Comparison of Entity Recognizers with Supervised Learning on CoNLL Dataset

Figure 5.26 shows the performance of the NERs based on the size of the training data. We evaluated the performance by continuously increasing the training set by ten documents (in total, CoNLL provides about 600 training documents). We can see that all NERs benefit from more training data. The dashed line signifies the performance of WebKnox Web (the NER in Web training mode). We used 1 to 50 seed entities per type and automatically generated training data using the algorithm from Figure 5.12. The upper x-axis shows the number of seeds that were used for training WebKnox Web. While this approach works without hand-labeled training data, the F1 value is about 29 % below the maximum F1 value of the best NER when using completely annotated training data. Still, even when LingPipe uses 20 training documents, the Web training approach still outperforms the NER, and even when OpenNLP's NER uses over 250 training documents, the WebKnox NER still performs better.

[27]`http://cogcomp.cs.illinois.edu/page/software_view/4`, last accessed on 18th of June 2012

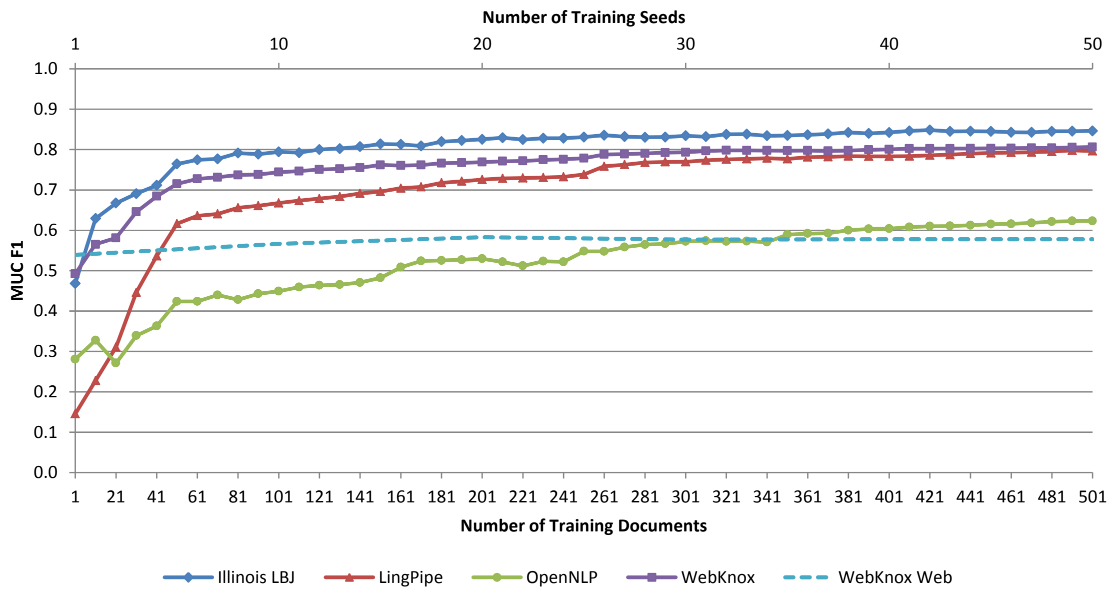


Figure 5.26: Comparison of Entity Recognizers based on the Amount of Training Data on CoNLL Dataset

Evaluation on TUD 2011

Figure 5.27 shows the comparison of the four NERs on the TUD data. All NERs were trained on 50 % training data and tested on the other 50 % test data from the dataset. It is clear that all NERs perform far worse on the TUD dataset than on the CoNLL dataset. The worsened performance stems from a smaller number of training documents, more diverse training data, and, most importantly, from the higher number of entity types that need to be recognized. From this chart, we can conclude that even state-of-the-art NERs perform poorly on diverse Web text when the task is to extract unseen entity mentions. The performance could be improved by allowing gazetteers, but this is not our focus.

Figure 5.28 shows the performance of the NERs based on the size of the training data from the TUD dataset (in total, the dataset provides about 60 training documents). We increased the number of training documents by five for each evaluation. We can see a similar behavior of the NERs as in Figure 5.26. We trained the WebKnox Web NER the same way as before, supplying it with automatically generated training data. It takes the Illinois LBJ NER about ten training documents to reach the performance of the WebKnox Web NER trained on 10 seeds. This time, however, the number of training documents is too small for LingPipe and OpenNLP to reach the automatically trained NER's performance at all. Training WebKnox Web with 50 seed entities per type performs only about 9 % worse than the Illinois LBJ after 56 training documents.

We can see a promising direction of automatically training the NER on the Web. We believe more research will be targeted in this direction since acquiring training data is extremely costly and when a new entity type should be recognized, the complete dataset needs to be manually re-annotated.

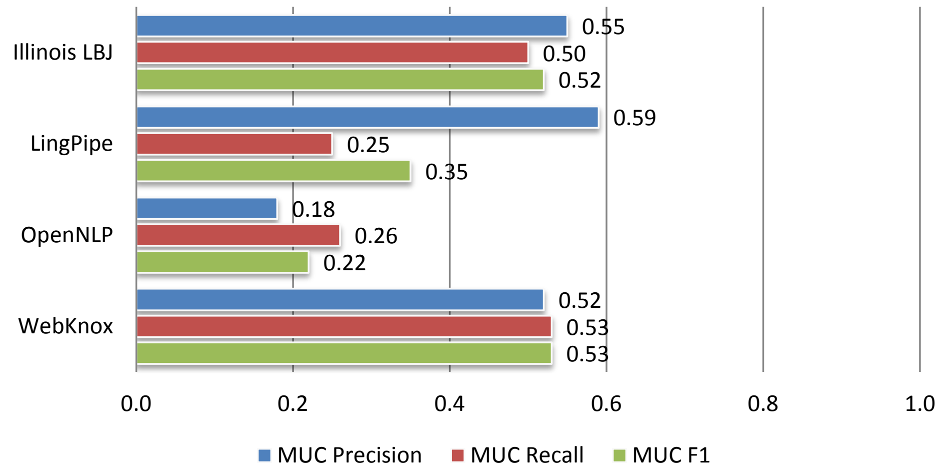


Figure 5.27: Comparison of Entity Recognizers with Supervised Learning on TUD Dataset

The experiments of this section have shown that our NER that was also used for the extraction of entities from plain text (NLPE in previous experiments) is comparable to the best NERs. We want to make two points in this conclusion. First, NER is far from being a solved problem. The low scores for NLPE in our entity extraction task on 17 concepts show that the best algorithms are not yet good enough to recognize and extract entities precisely – at least not without large dictionaries of known entities. Second, we have shown that we can train a named entity recognizer on sparsely-labeled training data which can be automatically generated. This approach is another step in solving the problem of generating training data for supervised machine learning approaches.

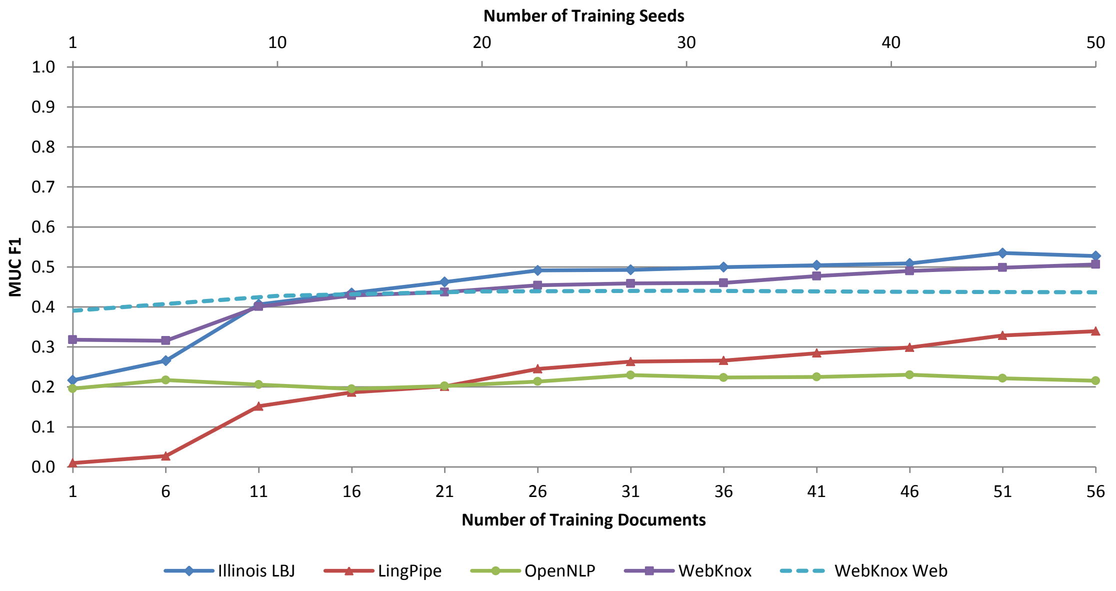


Figure 5.28: Comparison of Entity Recognizers based on the Amount of Training Data on TUD Dataset

5.4 Summary

This chapter reviewed techniques to extract entities from the Web. First, we classified different approaches for entity extraction and presented related work. After reviewing the related work, we have not found any research comparing different extraction techniques regarding their precision or number of extractions. We close this gap by presenting five extraction techniques and comparing their extraction precision and number of extractions on a set of 17 concepts. We found that there are large differences in extraction precision, but in the end, all the techniques should be used since none of the techniques find all entities that the other techniques find. It is also noteworthy that the average extraction precision of about 45 % is much too low to create a knowledge base with reliable information about entities. In the next chapter we will present methods to assess the extracted entities and raise the ratio of correct entities in our knowledge base.

Chapter 6

Assessment of Extractions

As seen in Chapter 5, extracting information from semi- or unstructured information sources is far from 100 % accurate. Information extraction systems extract the desired information from these sources under certain assumptions that do not always hold true. Every extraction technique has its own shortcomings and is error prone in some sense. This makes it necessary to assess the extracted information to ensure high precision. In this section, we show existing techniques to score and assess extracted named entities. The assessment of extracted facts is another interesting and important topic, but it is beyond the scope of this thesis. Please refer to Wu and Marian (2007) and Urbansky et al. (2008) for further reading on assessing extracted facts.

In this chapter, we give a brief overview of the related work, design our own assessment algorithms, and compare our algorithms to state-of-the-art techniques. In Thesis 3 in Section 1.4, we state that we can use a novel set of features for machine learning for the entity assessment problem to reach a higher precision than comparable techniques. We will evaluate this thesis at the end of this chapter.

6.1 Related Work

We group the assessment techniques into five main techniques: syntax, dictionary, redundancy, co-occurrence, and graph-based. We review the work most related to the focus of this thesis and discuss its applicability to our problem.

6.1.1 Syntax-based

Agrawal et al. (2009) study the problem of enhancing Web search results with structured information from databases. For example, a user might search for “small, fast, and handy digital cameras” and retrieve a list of documents from the search engine. Agrawal et al. (2009) developed a system that automatically extracts entity mentions from the first couple of search results, classifies them as true or false mentions, and looks for information about those entities in structured databases. The user in our example would therefore not only get a list of documents that mention small, fast, and handy digital cameras, but also concrete instances, such as the *Sony Cybershot*. Their system is related to our goals because they perform an entity extraction and classification step. The entity extraction step, as they call it, is more a recognition step since they simply compare entities from the database with phrases in the documents on a purely syntactic level. For long, specific entity names, such as the product name *Sony Cybershot DSC-hx9v*, this matching strategy is likely to recognize true mentions of the entity in the documents. For movie titles such as *Pi*, *Avatar*, or *300*, it is more complicated since these movie names can also be used as general terms in documents and may not refer to the entity. Therefore, the entity mention classification step is needed. The entity mention classification uses a linear support vector machine to classify an entity mention as true or false. Agrawal et al. (2009) train the SVM with four types of features:

1. **Document Level**: The type of entity often depends on the content of the whole document in which it appears. Agrawal et al. (2009) train the classifier with a set of 30,000 n-gram features with $0 < n < 4$.
2. **Entity Context**: The context of the entity mention is the text just before and right after the entity mention. These context words give strong clues about the entity type. For example, the preceding context words “directed the movie” would strongly hint that the following entity is a movie. In their system, they learn the 10,000 most frequent n-grams for both the preceding and following contexts of the entity mention.
3. **Entity Token Frequency**: This feature is the prior probability that an entity name really refers to an entity and not to a general phrase. For example, they train the system to recognize that the movie *300* has many false mentions (used as the number 300 and not the movie). Thus, the system is less likely to classify a mention of “300” as a movie compared with the mention *No Country for Old Men*, which almost always refers to the movie.
4. **Entity Token Length**: The number of words in an entity mention is another indicator for the classifier. Short entity mentions are more likely to be generic phrases (for example, *Avatar* compared to longer mentions such as *No Country for Old Men*).

In their evaluation, Agrawal et al. (2009) have shown that their entity classifier works with an accuracy of 97.2 % when tested on 100,000 movie entities. Their approach might be applicable in our system as well. The basic problem, however, is different from ours – they only recognize already known entities from a database and classify whether their recognition worked correctly. In this thesis, we study the problem of entity discovery, that is, finding unknown instances of entities and assessing our extractions.

Lin and Hung (2002) also use the left and right contexts of the named entity candidates in addition to the entity candidate’s structure to verify it. Instead of just learning the assessor model from positive examples, they use negative examples as well, and compute the probability that the candidate is correct. According to their evaluation on Chinese person names, their probabilistic approach led to an increase in F1 value of about 8 %.

Furthermore, Florian (2002) and Florian et al. (2003) have shown that a combination of classifiers (robust linear classifier, maximum entropy, transformation-based learning, and hidden Markov models) results in better overall classification performance.

6.1.2 Dictionary-based

Another way to check the validity of extracted entities is to look them up in dictionaries, databases, gazetteers, or knowledge bases. This technique is often an integral part of state-of-the-art NER systems. After the delimitation phase, NERs need to classify the entity candidates. Recognition can be supported by a source of known entities which can be queried to look up entities (see Section 5.1.5 for a detailed explanation). Similarly, these lists can help when assessing known entities.

Tanaka (2008) tries to improve Web search by re-ranking information based on search engine hit counts and number of social bookmarks. The Social Bookmark Rank (SBRank) can be used to look up the word "Java", for example, and count the number of social bookmarks.

This assessment technique is less relevant to our work since we focus on discovering unknown entities, which we could not look up in a list.

6.1.3 Redundancy-based

One of the simplest approaches to assess extracted entities is to apply the hypothesis that "extractions drawn more frequently from distinct sentences in a corpus are more likely to be correct." (Downey, 2008). One could determine a frequency threshold and only classify entities that are extracted more frequently than the threshold as correct. The problem is, however, to determine this threshold which calls for more sophisticated models based on redundancy.

Riloff and Jones (1999) use automatically learned extraction patterns and rank extracted entities by the number of distinct patterns used to find them. Instead of relying on the frequency of the extraction, they also take the extraction type into account. They do not, however, calculate any kind of threshold to separate the correct and incorrect extractions.

The "URNS model" (Downey et al., 2005, Downey, 2008) is a redundancy-based, unsupervised approach to determine whether an entity extraction is correct or not. It uses the classic, probabilistic "balls and urns" model from combinatorics. In this model, each extraction is equal to a labeled ball that is put into the urn. The label can either be a correct entity extraction for a concept or an error. URNS relies on redundancy on the Web and therefore assumes that correct labels appear on several balls in the urn. For example, the entity *Jim Carrey* for the actor concept might have been extracted on several pages, which supports the belief that *Jim Carrey* is really an actor. *Jim Carrey* then becomes a correct label found on several balls in the urn.

The redundancy is modeled as a monotonic feature, that is, a feature that with increasing value (all other feature values being equal) increases probability of concept membership. For classifying named entities, there must be more correct extractions than incorrect extractions in the urn, otherwise the model will provide inaccurate probability estimates (Downey, 2008). This requirement limits the applicability of URNS for assessing extracted named entities from the Web since there are many entities that appear only sparsely and can therefore not be assessed correctly using this model.

The extraction process is then handled as repeated draws from the urn with replacement.

The question that URNS tries to answer is: "given that a particular label x was extracted k times in a set of n draws from the urn, what is the probability that $x \in C$ [C is the set of correct labels]?" (Downey et al., 2005), or more formally as shown in Equation 6.1 (Downey et al., 2010).

$$P(x \in C \,|\, x \text{ appears } k \text{ times in } n \text{ draws}) = \frac{P(x \text{ appears } k \text{ times in } n \text{ draws} \,|\, x \in C) \times P(x \in C)}{P(x \text{ appears } k \text{ times in } n \text{ draws})} \tag{6.1}$$

In information extraction, different patterns and extraction techniques are often applied that have different failure rates. For example, to extract entities for the concept *Actor* the two patterns "actors such as" and "stars in this movie" could be used. The entities that are extracted by the first pattern might be correct more often than the ones extracted by the latter pattern. Moreover, entities that have been extracted with multiple patterns are more likely to be correct than entities that only appear in one. To model this additional knowledge, URNS can be generalized using multiple urns where each urn stands for a different extraction technique or pattern (Downey et al., 2005).

URNS can be seen as a binary classifier that assigns true or false to an extraction after estimating the likelihood that it belongs to a certain concept. Other classification models can also be applied for that task (such as support vector machines, noisy-or, or Bayes Classifiers), but experiments have shown that URNS performs better than comparable techniques (Downey et al., 2005).

6.1.4 Co-occurrence-based

The techniques described next make use of co-occurrence statistics between the entity and its type.

Latent Relational Analysis

Latent relational analysis (LRA) is a technique to measure the semantic similarity between word pairs (Turney, 2005). Two pairs are called analogous when they have a high semantic similarity. For example, the two word pairs "airplane:flight" and "car:drive" are very similar; one could say "airplane" is to "flight" as "car" is to "drive". This measurement can now also be used to assess entity extractions. Nadeau (2007) calls his technique "Statistical Semantics Filter" and it uses word pairs consisting of the assigned concept and the extracted entity (`CONCEPT:ENTITY`). For example, "City:London", "City:Dublin", "Actor:Bruce Willis", and "City:Click Here" are such pairs. The semantic similarity between the pairs "City:London" and "City:Dublin" is very high (both are correct extractions), whereas the similarity between "City:London" and "City:Click Here" is very low (since "Click Here" is not a correct extraction for the concept *City*). To calculate the semantic similarity between word pairs, a matrix of word pairs and extraction patterns is built. An extraction pattern for this algorithm is a regular expression that finds the entity and the concept in a text with a maximum of three intervening words. For example, if the passage "in the city of London" is found in a text, the pattern "city * London" is created. There can be several extraction patterns for one word pair.

An example matrix is shown in Table 6.1. The bold word pairs are seeds that are known to be correct. The entries in the matrix represent how frequently a word pair was found using an extraction pattern. In the algorithm from Nadeau (2007) these values are smoothed using entropy and log transformations. The matrix will contain many zero values and is compressed using singular value decomposition (SVD).

Word Pair	**"city * CITY"**	**"CITY * * city"**
City:London	10	50
City:Dublin	8	66
City:New York	12	55
City:Click Here	5	2

Table 6.1: Example LRA Co-occurrence Matrix

The scores for the extracted word pairs are now determined by calculating the cosine similarity of the row vector of the word pairs with the average vector from all seed word pairs. A score threshold is determined by taking the smallest cosine similarity of all seed vectors. If the resulting similarity is bigger than the threshold, the extraction is considered to be correct; if it is not, the extraction will be considered an error. Equations 6.2 – 6.6 show the calculations for the example from Table 6.1. We can see that the correct extraction "City:New York" has a score above the threshold and is therefore considered to be a correct extraction. The incorrect extraction "City:Click Here" yields a low semantic similarity with the average seed vector and is therefore discarded. Equations 6.2 and following show how the similarities are calculated for the example from Table 6.1.

$$Average\vec{Seed}Vector = \langle (10+8)/2, (50+66)/2 \rangle = \langle 9, 58 \rangle \tag{6.2}$$

$$sim(\vec{a},\vec{b}) = cos(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{||\vec{a}|| \times ||\vec{b}||} \tag{6.3}$$

$$ScoreThreshold = sim(\langle 10, 50 \rangle, \langle 8, 66 \rangle) = 0.997 \tag{6.4}$$

$$score(City : NewYork) = sim(\langle 12, 55 \rangle, Average\vec{Seed}Vector) = 0.998 \tag{6.5}$$

$$score(City : ClickHere) = sim(\langle 5, 2 \rangle, Average\vec{Seed}Vector) = 0.509 \tag{6.6}$$

In experiments LRA has shown slight but statistically significant improvements on named entity assessment (Nadeau, 2007). However, due to the many search queries that have to be generated, it is not the first choice for assessing named entity extractions.

Pointwise Mutual Information

Pointwise mutual information (PMI) (Church and Hanks, 1990) is a statistical technique to calculate how often two words or phrases occur together in a document collection. PMI was first used by Turney (2001) in his PMI-IR algorithm. Etzioni et al. (2004) use patterns as discriminators to ensure the correctness of an extracted fact or entity. This means they use these discriminators as queries for Web search engines and calculate pointwise mutual information between the extraction and the discriminator with the hit counts. If E is an extracted entity and D is the discriminator phrase, the PMI can be calculated as shown in Equation 6.7.

$$PMI = \frac{Hits(E + D)}{Hits(E)} \tag{6.7}$$

For example, the PMI for the discriminator phrase "Jim Carrey is an actor" and the entity *Jim Carrey* can be used to calculate the probability that the discriminator phrase and the entity are found together on a Web page. Even for correctly extracted entities, this number is very small, perhaps 1:100,000. Using several automatically generated discriminators and the entity name leads to many PMI values that are treated as features for a naïve Bayes classifier. A set of k positive examples and k negative examples for each concept is used to automatically find the threshold to determine whether the entity really belongs to the concept or not (Yates, 2007).

There are several disadvantages of PMI for assessing named entity extractions. First, the seeds used for the threshold determination tend not to be representative of the whole set of extractions. In combination with the probability polarization behavior of the naïve Bayes classifier, it tends to estimate probabilities incorrectly (Downey, 2008). Second, this approach has more problems with classifying actually correct entities as incorrect (false negatives) than with classifying incorrect entities as correct (false positives) (Yates, 2007). This is because some entities (such as obscure movies) have low hit counts and always fall below the automatically determined threshold. Third, computing the PMI scores is expensive (time and computing resources) since there are two queries to a search engine for each extracted named entity that should be assessed.

REALM

The REALM system uses hidden Markov models (Rabiner, 1990) and n-gram-based (Manning and Schütze, 2002) language models to rank candidate extractions by the likelihood that they are correct (Downey, 2008). Unlike URNS, REALM can also be used to assess sparse extractions. An extraction for the REALM system is a relation consisting of two or more named entities. For example, *Headquartered(Google, Mountain View)* is such a relation where the company *Google* is located in *Mountain View*. The goal of REALM is to assess whether the relation holds true. It uses a set of automatically extracted seeds to learn the models for the assessment phases. The system performs two assessment steps separately. First, it checks the types of the relation arguments, that is, it checks whether *Google* is really a company and whether *Mountain View* is really a location. Second, it determines whether the relation

Headquartered holds for the two arguments. Before the two steps are performed, the LEX method (Downey et al., 2007) is used to recognize proper nouns and to combine multi-token proper nouns (such as *Los Angeles*) into a single token.

Type Checking The first step is type checking, which is done using an HMM. It exploits the distributional hypothesis, that is, it assumes that correct seed entities are likely to appear in similar contexts as the entity that should be assessed (Downey, 2008). The HMM is constructed for the seed entities and for the unknown entities. The more similar the state distributions $P(t \,|\, e_i)$, where t is the step in the HMM and e_i is the entity, between the seed and the unknown entity, the more likely is it that the assessed entity is correct. For example, the two extracted phrases *Chicago, Illinois* (seed) and *Pickerington, Ohio* (to be assessed) are given. While *Pickerington* is a very sparse entity (it might even appear only once in the corpus), the state distributions for *Illinois* and *Ohio* are quite similar, so *Pickerington* is likely to be assessed as a correct type (Downey, 2008).

Relation Assessment The second step is to assess whether the relation of the two entities is correct. Downey (2008) calls the component for this step REL-GRAMS, which again relies on the distributional hypothesis. All contexts around seed entities in the given corpus are collected. A context consists of n words around the seed entity. For example, in the sentence "companies such as Google are headquartered in" the following contexts ($n = 3$) are collected for the seed entity *Google*: "such as Google", "as Google are" and "Google are headquartered". All contexts from all seed entities form a set C. For any extracted relation, the contexts are extracted in the same manner and a context vector $v_{(e_j,e_k)}$ with the dimension of $|C|$ is built, where the i^{th} dimension is the number of times the context c_i occurred for the two entities e_j and e_k (Downey, 2008). This vector is then compared to a seed vector using a distance metric. The lower the distance, the more similar the contexts, and therefore the more likely that the relation holds true for the two given entities.

REALM ranks candidate extractions instead of classifying them as correct or incorrect. This is a limitation since most real systems need to know whether an entity is correct or incorrect instead of being presented with a probability of the entity being correct.

Wisdom of the Masses

Zhou et al. (2007) present an entity classification method that assigns a type to the entity depending on how the top-ranked Web pages refer to the entity. They send the entity name to a search engine, retrieve the top-ranked documents, extract the context around the entity mention, build a feature vector with features such as term and document frequency, and classify the entity using these features. The taxonomy of possible entity types has to be modeled before using this technique. A demonstration[1] of the system is available online.

[1]`http://dragon.ischool.drexel.edu/cptc.asp`, last accessed on 23rd of March 2012

6.1.5 Graph-based

Wang and Cohen (2007) build a graph of websites, wrappers, and extracted entities in order to rank the list of extractions with a random graph walk (RGW) algorithm. Figure 6.1 shows such an example graph.

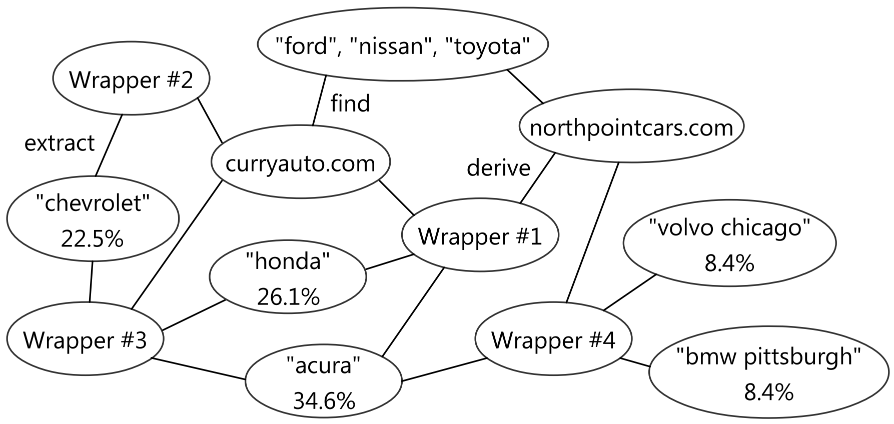


Figure 6.1: Example Graph of Websites, Extractions, and Wrappers (Wang and Cohen, 2007)

They build a transition matrix for the random walker. The transition matrix is a square matrix with nodes for Web pages, entities, and wrappers. Equation 6.8 shows how the transition probabilities are calculated. For example, the transition probability from node `curryauto.com` to node `Wrapper 1` in Figure 6.1 would be $\frac{1}{4}$ since there are four edges from and to `curryauto.com`.

$$M_{xy} = P(y \mid x) = \frac{1}{|x \rightarrow y|} \tag{6.8}$$

The probability vector $\vec{v}$ stores the probabilities for all nodes at certain time steps t. Equation 6.9 shows how the probability vector is updated iteratively. The probability vector is initialized with seeds in the beginning ($\vec{v}_0$). Positive seeds get a probability of 1, negative seeds get a probability of 0, and every unknown node is initialized with a probability value of 0.5. The parameter γ is used to restart the random walk. Wang and Cohen (2007) use a value of 0.5 to put more weight on the seeds.

$$\vec{v}_{t+1} = \gamma \times \vec{v_0} + (1-\gamma) \times M_{xy}\vec{v}_t \tag{6.9}$$

After some iterations (the number depends on the size of the graph), the probabilities for the nodes converge. It is now possible to rank the nodes by their scores. A high score on an extraction node indicates that the extraction was correct, while a lower score indicates that the extraction failed. For the example from Figure 6.1, that would mean we would get a sorted list of extractions with their scores from `acura` (score: 34.6 %) to `volvo chicago` (score: 8.4 %). Wang and Cohen (2007) use this ranking to show the extractions in this order to the user. To further process extracted named entities, it might be necessary to classify them as correct or incorrect, which means we need a threshold to decide whether an entity is correct.

TrustRank (Gyöngyi et al., 2004) is a graph-based algorithm that aims to classify pages as "good" or as "spam". The algorithm can then be used to rank search results from general purpose search engines such as Google. TrustRank uses the Web's link structure to construct a huge graph where each node is a Web page and each edge is a link between the pages. This graph is represented in a transition matrix. Each page is assigned a probability indicating how likely it is that the page is "good". These probabilities are stored in a probability vector. This vector is initialized with "good" (score 1) and "spam" (score 0) seed pages, and is updated iteratively using the transition matrix. Trusted pages spread their score to the pages to which they link. Thus, these pages are ranked more trustworthy. After a certain amount of iterations, the spam pages should have a low score (since few trustworthy pages link to them) and can be sorted out of the search results. In the next section, we will adopt this idea and present how we can use a similar algorithm to assess extracted entities.

6.1.6 Combination: Redundancy- and Syntax-based

Nadeau (2007) studied the value of syntactical features for assessing entities, and combined a frequency-based noise filter (URNS) with a lexical noise filter. Both assessment techniques assign probability estimates for an entity extraction. The final estimate for a combined assessor is the average of the both estimates. Nadeau's hypothesis was that entities within the same concept must have similar lexical features. For example, person names usually consist of two words where the first word is relatively short. Company names, on the other hand, can be much longer, contain special characters, have different capitalization, et cetera. His noise filter system uses a list of 50 features, including text length, number of words, number of white spaces, and so on. The features are all drawn from a commonly used pool of features in NER as shown in Table 5.1. He used positive and negative examples and the SMOTE one-class learner (Chawla et al., 2002) to build a model for a classifier.

6.1.7 Summary of Related Work

In this section, we have grouped related work on assessment of extraction results into five main groups, namely syntax, dictionary, redundancy, co-occurrence, and graph-based assessment methods. Combinations of several methods can also be used, but there is still little insight on which combinations are suited for the task of assessing extracted entities. In this thesis, we have a scenario where we extract entities using different extraction techniques applied to different sources on the Web. In this scenario, we are able to extract a large set of features, which can be used to determine the correctness of an extraction. We will therefore build an entity extraction assessor that combines several assessment methods and compare it to the methods from the related work.

6.2 Modifying Assessment Techniques for Assessing Extracted Entities

We will now build several entity assessment algorithms based on the ideas presented in the related work section and compare them with each other. The goal is to find out which assessment technique works best for our set of extractions across the 17 concepts. In the following sections, we will explain how we modified and implemented each algorithm before we compare them in the last section.

Each assessment strategy has to instantiate Equation 6.10, that is, given an *entity*, a *concept* it supposedly belongs to, and *context* information, the assessment strategy has to determine whether the entity is a correct or incorrect extraction for that concept. The *context* can hold meta information that is needed by the assessment algorithm, which differs in each approach. We provide an instantiation of this equation for each of the assessment strategies that we evaluate in this thesis.

$$assess(entity, concept, context) = \begin{cases} correct \\ incorrect \end{cases} \tag{6.10}$$

6.2.1 Pointwise Mutual Information

We explained the basic principles of this approach in Section 6.1.4. In this section, we provide more details on how we modified PMI for the evaluation. We implemented it very closely according to the descriptions given by Yates (2007), Etzioni et al. (2004) and Soderland et al. (2004) and use the following discriminator patterns, where `CS` is the singular of the concept name (for example *Car*), `CP` is the plural of the concept name (for example *Cars*), and `X` is the name of the entity that should be assessed (for example *Fiat Evo*). One complete discriminator phrase could be "Cars such as Fiat Evo", which is then used to query a search engine (using phrase matching).

```
"CP such as X"
"such CP as X"
"CP like X"
"CP especially X"
"CP including X"
"X and other CP"
"X or other CP"
"X is a CS"
"X is the CS"
```

PMI uses training and tuning entities for each concept, which we denote with *trainingSet*. We use the same number of positive and negative labeled entities for training. Negative instances are entities that were extracted for the concept, but marked as incorrect. We could also use correct entities from other concepts. We now calculate the PMI values for all discriminator phrases *disc* and all entities in the *trainingSet*. Equation 6.11 shows how we calculate the average positive and average negative PMI per concept using training sets. Positive training entities should have higher PMI values than negative training entities.

$$AveragePMI_{+/-}(trainingSet_{+/-}, disc) = \frac{\sum_{d \in disc} \sum_{e \in trainingSet_{+/-}} PMI(e, d)}{|trainingSet_{+/-}|} \tag{6.11}$$

We now search for a threshold to separate positive and negative instances. This threshold is different for each *concept* and is determined by Equation 6.12. For each *concept* we have *trainingSet* and *disc* to calculate the average PMI for positive and negative instances as shown in Equation 6.11, but for brevity we left out these arguments in Equation 6.12.

$$threshold(concept) = AveragePMI_{-} + \frac{AveragePMI_{+} - AveragePMI_{-}}{2} \tag{6.12}$$

After determining the threshold for each concept using training entities, we use a subset $tuningSet \subset trainingSet$ with $tuningSet \cap (trainingSet \setminus tuningSet) = \emptyset$ of k positive and k negative entities. We then calculate the following four *probabilities* for each concept, where e is the entity, d is the discriminator pattern, and *threshold* is the PMI threshold that we determined for the given concept:

1. $P(PMI(e, d) > threshold \,|\, +)$
2. $P(PMI(e, d) > threshold \,|\, -)$
3. $P(PMI(e, d) \leq threshold \,|\, +)$
4. $P(PMI(e, d) \leq threshold \,|\, -)$

We can now use the learned probabilities and naïve Bayes classifier to calculate the probability that an entity belongs to the concept. Equation 6.13 now instantiates our target Equation 6.10, where *context* holds information about the learned *probabilities* which are then used in the *classify* function of the naïve Bayes classifier.

$$assess_{PMI}(entity, concept, context) = \begin{cases} correct & \text{if } classify(probabilities) > 0.5 \\ incorrect & \text{if } classify(probabilities) \leq 0.5 \end{cases} \tag{6.13}$$

6.2.2 Latent Relational Analysis

The second co-occurrence-based assessor we evaluate is based on latent relational analysis and works as described in Section 6.1.4. Similar to PMI, LRA is based on frequencies of discriminator phrases. We use the same set of phrases that we used for PMI as shown in Section 6.2.1.

For each concept, we calculate an $average\vec{Seed}Vector$ and a *scoreThreshold* (see Section 6.1.4).

Equation 6.14 instantiates our target Equation 6.10, where *context* holds information about the *scoreThreshold* we calculated for each *concept*.

$$assess_{LRA}(entity, concept, context) = \begin{cases} correct & \text{if } score(concept : entity) > scoreThreshold \\ incorrect & \text{if } score(concept : entity) \leq scoreThreshold \end{cases} \tag{6.14}$$

6.2.3 Random Graph Walk

Our graph walk assessment algorithm uses an idea similar to the TrustRank presented in Section 6.1.5. Figure 6.2 shows a sample graph consisting of extracted entities (rectangles), the Web pages from which they have been extracted (rounded boxes), and the techniques used for extraction (triangles).

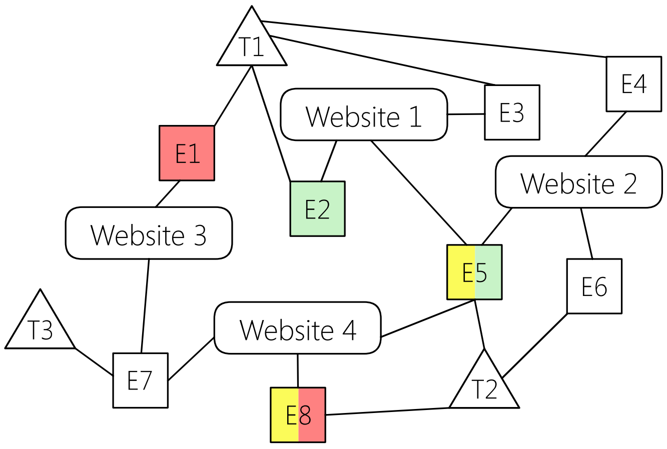


Figure 6.2: Example Graph for Graph-based Entity Assessment

We divide our training set into two smaller sets. One set contains “seed entities”, that is, entities for which we know whether they were correctly extracted (compare with spam page or authority page in TrustRank). The other set consists of “control entities”, that is, entities for which we know whether they belong to the concept but treat them during the random walks as if we did not know. In Figure 6.2, the positive seed entity is `E2` (green) and the negative seed is `E1` (red). The control entities in the sample graph are the correct entity `E5` (yellow-green) and the incorrect entity `E8` (yellow-red).

Now we build a transition matrix for the random walker. The transition matrix is a square matrix with nodes for Web pages, entities, and extraction techniques. Equation 6.15 shows how the transition probabilities are calculated, where x is the source node and y is the target node. For example, the transition probability from node `E7` to node `Website 3` in Figure 6.2 would be $\frac{1}{3}$ since there are three edges from and to `E7`.

$$M_{xy} = P(y \mid x) = \frac{1}{|x \to y|} \tag{6.15}$$

Equation 6.16 shows how the vector $\vec{v}$ is updated iteratively (compare to Equation 6.9). It is important to note that the transition probability matrix M_{xy} is transposed. The transposition means that the scores of neighbor nodes are now distributed by their number of outgoing edges, which has the effect that we no longer have probabilities at the nodes, but scores instead. For example, in Figure 6.2, `Website 1` does now get $\frac{1}{3}$ from `E2`, `E5`, and `E3` instead of $\frac{1}{2}$ from `E2`, $\frac{1}{4}$ from `E5`, and $\frac{1}{2}$ from `E3`.

$$\vec{v}_{t+1} = \gamma \times \vec{v_0} + (1-\gamma) \times {M_{xy}}^{\top} \vec{v}_t \tag{6.16}$$

After some iterations (the number depends on the size of the graph), the scores for the nodes converge. We now calculate the average trust of all positive control entities and negative control entities to determine the threshold for assessing the unknown entities. Equation 6.17 shows how the trust threshold is calculated, where $C+$ is the set of positive control entities and $C-$ is the set of negative control entities. The parameter δ is used to shift the threshold toward the negative ($\delta < 0.5$) or toward the positive ($\delta > 0.5$) threshold boundary and therefore we can make the precision-recall trade-off. We calculate one threshold per concept.

$$\mathit{threshold} = (1-\delta) \times \frac{\sum_{c \in C-} \vec{v}(c)}{|C-|} + \delta \times \frac{\sum_{c \in C+} \vec{v}(c)}{|C+|} \tag{6.17}$$

Equation 6.18 is now an instantiation of our target Equation 6.10, where *context* holds information about the score vector $\vec{v}$ and *threshold* we calculated for each *concept*.

$$\mathit{assess}_{RGW}(\mathit{entity}, \mathit{concept}, \mathit{context}) = \begin{cases} \mathit{correct} & \text{if score of node } \mathit{entity} \text{ in } \vec{v} > \mathit{threshold} \\ \mathit{incorrect} & \text{if score of node } \mathit{entity} \text{ in } \vec{v} \leq \mathit{threshold} \end{cases} \tag{6.18}$$

6.2.4 Text Classification

The text classification assessor uses only character level n-grams of the entity name itself for assessment. Intuitively, entities within one category often have repeating character sequences. For example, in car names, the manufacturer is repeated (for example, *Ford Mustang*, *Ford Shelby*). The text classifier is supposed to find these sequences and classify entities accordingly. For each concept, we learn one binary text classifier using correct extractions and incorrect extractions. See Section 5.2.4 for more detail on how the text classifier functions. Equation 6.19 shows the instantiation of the target Equation 6.10.

$$\begin{gathered} \mathit{assess}_{text}(\mathit{entity}, \mathit{concept}, \mathit{context}) = \begin{cases} \mathit{correct} & \text{if } \arg\max_{\mathit{type} \in T} S(\mathit{type} \mid \mathit{candidate}) = + \\ \mathit{incorrect} & \text{if } \arg\max_{\mathit{type} \in T} S(\mathit{type} \mid \mathit{candidate}) = - \end{cases} \\ \text{where} \\ S(\mathit{type} \mid \mathit{candidate}) = \sum_{n \in N_{candidate}} \mathit{relevance}(n, \mathit{type}) \end{gathered} \tag{6.19}$$

6.2.5 Feature-based Assessor

We have more information about the extracted entities than we use in previous assessment strategies. This assessor tries to use as many relevant features as possible to improve assessment performance. We found a set of seven of relevant features to train and use a naïve Bayes (NB) classifier and a k-nearest neighbor (KNN) classifier. We tried using more features, such as the number of sources from which the entity has been extracted, but not all features improved the classification accuracy. Our methodology was adding feature by feature, keeping a feature if it improved the classification accuracy, and dropping it if it did not (this approach assumes feature independence but decreases the search space tremendously).

1. **Number of Concepts**: The number of concepts to which the extraction has been assigned, for instance, the extraction *Australia* could belong to the concept *Country* and *Movie*.
2. **Word Count**: The number of words belonging to an entity.
3. **Special Character Count**: The number of non-alphanumeric characters in the entity name, such as `-` or `:`.
4. **Digit Start**: Whether the entity name starts with a digit.
5. **Uppercase**: Whether the entity name starts with an uppercase letter.
6. **Trusted Source**: Whether the entity was extracted from a Web page from which one of the trusted entities was also extracted.
7. **Syntactical Similarity**: The idea behind this feature is that entities of the same type often contain similar words, for example, entities of the *Mobile Phone* concept often contain the manufacturer name, such as *Nokia* or *Samsung*. An unknown entity that contains similar words as the seed entities is therefore more likely to be correct. We therefore compare the candidate to all known positive entities.

Equation 6.20 is the instantiation of the target Equation 6.10 for the naïve Bayes classifier. The *context* contains information about the sources from which the entity and positive seed entities were extracted and the positive seed entities themselves.

$$assess_{NB}(entity, concept, context) = \begin{cases} correct & \text{if } P(+ \mid entity, concept) > P(- \mid entity, concept) \\ incorrect & \text{if } P(+ \mid entity, concept) \leq P(- \mid entity, concept) \end{cases} \tag{6.20}$$

Equation 6.21 is the instantiation of the target Equation 6.10 for the KNN classifier, where *context* is all training instances for the given concept. We use a Euclidean distance function for the classifier with $k = 3$.

$$assess_{KNN}(entity, concept, context) = \begin{cases} correct & \text{if the majority of neighbors are correct} \\ incorrect & \text{if the majority of neighbors are incorrect} \end{cases} \quad (6.21)$$

The naïve Bayes classifier uses all seven features listed above. For the k-Nearest Neighbor it has been shown that using only the syntactical similarity feature yields the best performance.

6.2.6 Combined Assessor

The combined assessor uses at least two independent assessors and classifies an entity as correct if all assessors independently classify the entity as correct. We have tried several combinations of existing assessors, if we write "combined" without further descriptions, we mean the combination of the random graph walk and the naïve Bayes assessor as we found this combination to be the best (the charts in Appendix B explain this finding in detail).

Equation 6.22 shows the assessment function for the RGW+NB combination. An entity is only classified correct if both assessors classify it independently as correct. Note that we left out the arguments *entity*, *concept*, and *context* for brevity.

$$assess_{Comb} = \begin{cases} correct & \text{if } assess_{RGW} \wedge assess_{NB} = correct \\ incorrect & \text{if } assess_{RGW} \vee assess_{NB} = incorrect \end{cases} \quad (6.22)$$

6.3 Evaluation of Entity Assessment Techniques

To evaluate the performance of the presented assessment strategies, we extracted over 6,000 entities from the Web using the extraction techniques described in Chapter 5. For each concept, we divided the data into training (including tuning if necessary) and test data (between 54 and 180 entities per concept). The training set and the test set each hold an equal number of positive and negative entities. As performance measures we use precision, recall, F1, and accuracy.

6.3.1 Dependency on the Amount of Training Data

In this experiment, we want to find out how much the performance of each assessor depends on the size of the training set. We ran all assessors 10 times using 10–100 % of the training data.

Each chart shows precision, recall, and F1 curves for the assessors. Additionally, we plotted an F1 threshold, which shows the F1 value for precision@recall=1. The test set is equally split into positive and negative entities. A recall of 1 would yield a precision of 0.5 and an F1 of $\frac{2}{3}rd$. An F1 value below this threshold therefore means that the assessor performs worse than classifying each entity as correct.

Figure 6.3 and Figure 6.4 show that both co-occurrence-based classifiers (PMI and LRA) are relatively independent of the amount of training data given. PMI has learned thresholds that are too high and therefore classifies only few extracted entities as correctly extracted. This high threshold yields a low recall which decreases the F1 value to under 20 % – below the F1 threshold. The LRA assessor seems to classify almost every entity as correct leaving out only very few instances and therefore the F1 value is close to the F1 threshold. We can conclude that the amount of training data does not play an important role for neither PMI nor LRA.

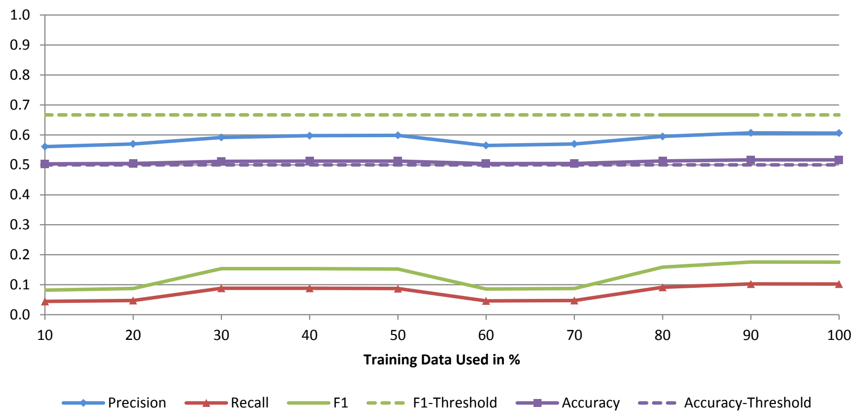


Figure 6.3: Performance of PMI, Dependent on the Amount of Training Data

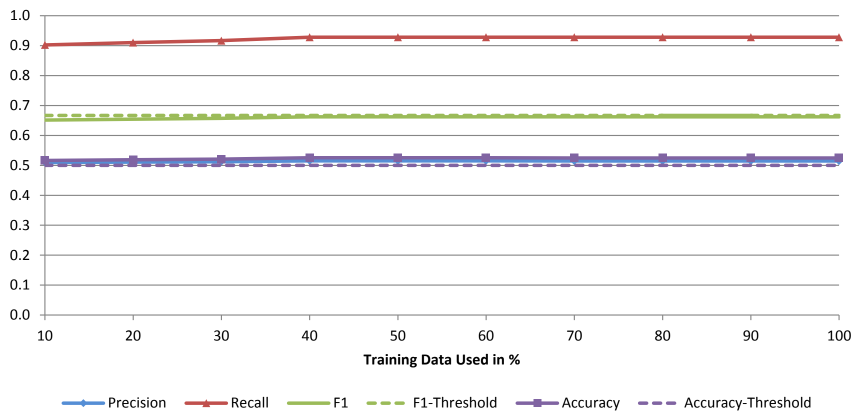


Figure 6.4: Performance of LRA, Dependent on the Amount Training Data

Figure 6.5 shows the performance of the Random Graph Walk assessor. This assessor is also largely independent of the amount of training data given. A few seed and tuning entities seem adequate to determine thresholds to separate correct and incorrect extractions. If we use more than 70 % of the training data, we can see that the accuracy, precision, and F1 value increase. Both F1 value and accuracy are above their thresholds when we use all available training data.

Figure 6.6 shows the performance of the text classification-based assessor dependent on the size of the training data. We can see that using more training data closes the gap between precision and recall scores. The precision decreases as the recall increases and at about 30 % of the training data, the precision equals the recall. The F1 value stays below the F1 threshold

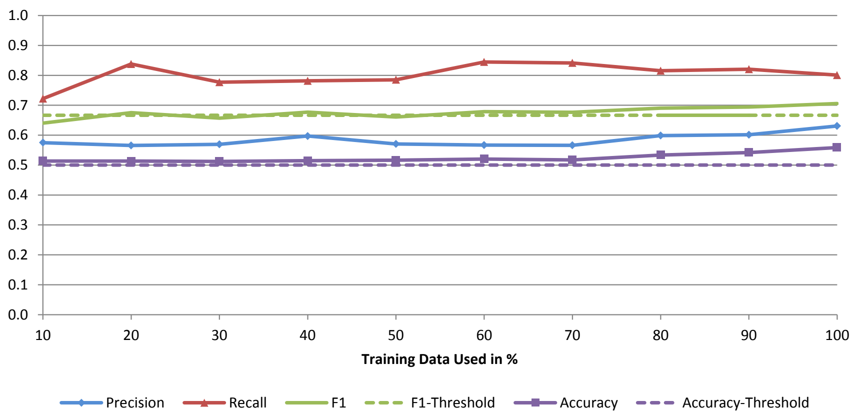


Figure 6.5: Performance of RGW, Dependent on the Amount of Training Data

all the time but the accuracy is about 12 % above the accuracy threshold, largely independent of the amount of training data used.

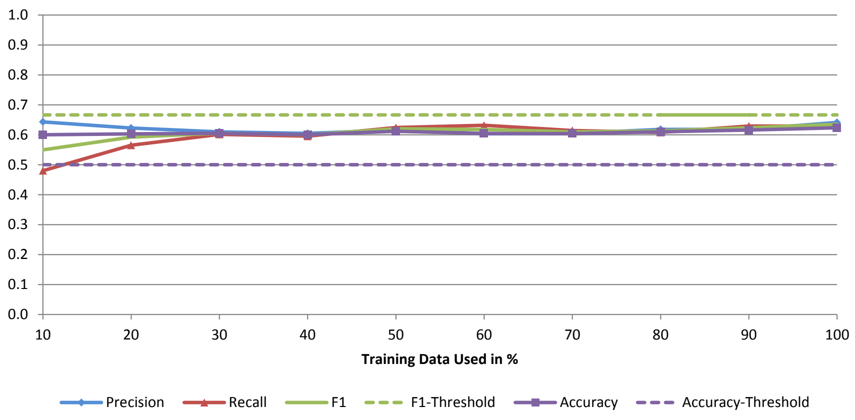


Figure 6.6: Performance of Text Classification, Dependent on the Amount of Training Data

Figure 6.7 shows the performance of the naïve Bayes assessor using seven features. Contrary to what we observed with the text classification assessor, the precision stays about the same with increasing training data and the recall rises slightly. For the seven features, the amount of training data could still be too little to successfully learn the conditional probabilities. Both F1 value and accuracy are above their respective thresholds.

In Figure 6.8, we see the performance of the KNN using only the syntactic similarity features. The assessor is also largely independent of the amount of training data used. The naïve Bayes assessor, however, clearly outperforms KNN. It is interesting though that using only few syntactic features yields a better performance than both co-occurrence-based approaches, the graph-based approach and the text-classification-based approach. Syntactic features are obtained faster than search engine hit scores (as needed for the co-occurrence-based assessors), for example. Further tests with combinations between the two classifiers did not yield performance improvements.

Figure 6.9 shows the performance of the combination of random graph walk and the naïve

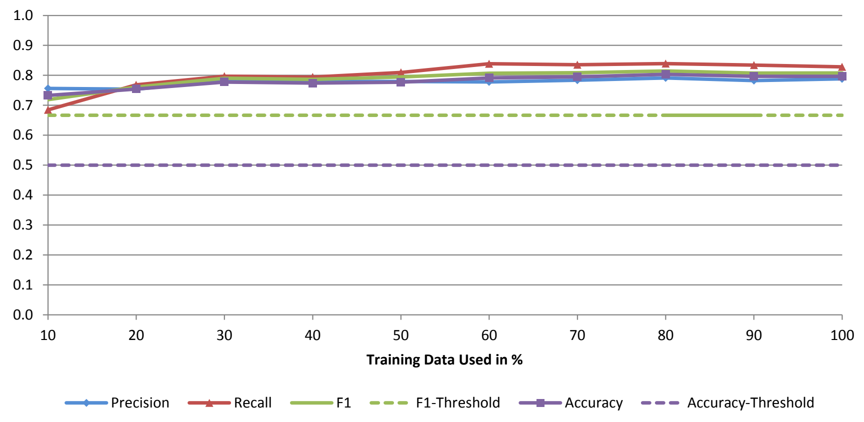


Figure 6.7: Performance of NB, Dependent on the Amount of Training Data

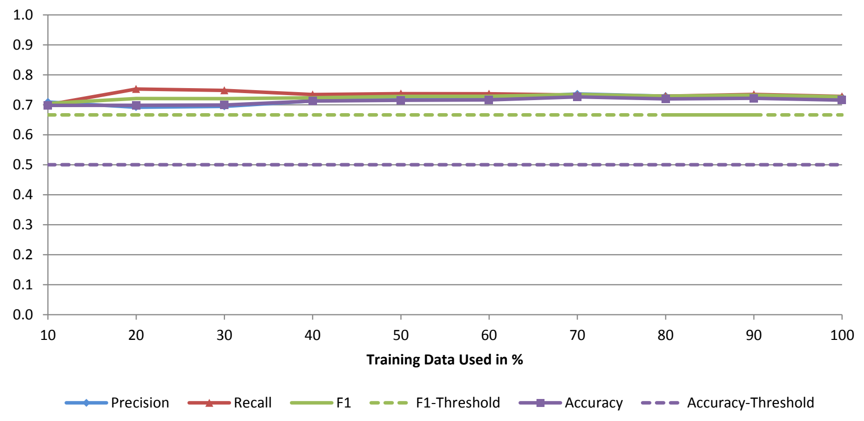


Figure 6.8: Performance of KNN, Dependent on the Amount of Training Data

Bayes dependent on the size of the training data. We can observe that the precision and accuracy rise more with increasing training data compared to random graph walk (compare Figure 6.5) or naïve Bayes alone (compare Figure 6.7). The combined assessor is slightly worse than the naïve Bayes assessor when comparing the F1 value and accuracy and using all available training data. The precision peak of the combined assessor, however, is about 5 % better than the one of the naïve Bayes assessor.

6.3.2 Assessment Techniques Performance by Concept

In this experiment, we want to find out how concept-dependent the assessment algorithms function. We use 100 % of the available training data for each assessor and report the precision, recall, and F1 value for each assessor and concept.

Figure 6.10(a) shows the performance of the PMI assessor for each concept. We can see that the average low recall and F1 value are due to the fact that PMI does not classify many entities across the different concepts as correct. Note that the precision for the concepts without a bar in the graphic is undefined, not zero. The precision is undefined because PMI never classified entities as correct. Recall and F1 value, on the other hand, are zero because

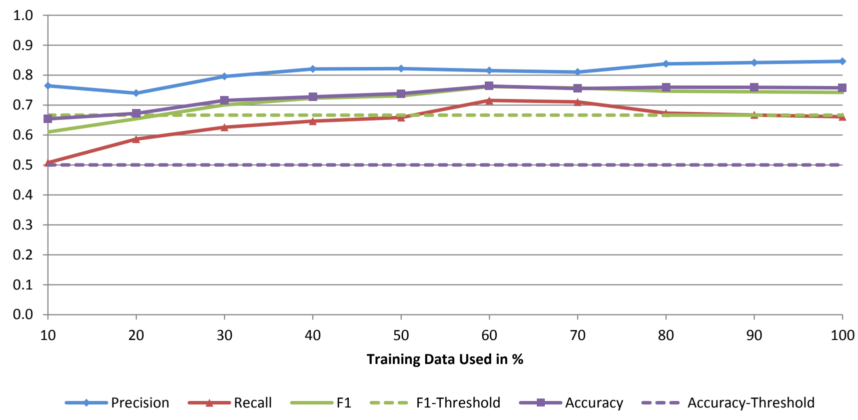


Figure 6.9: Performance of the Combined Assessor, Dependent on the Amount of Training Data

in each concept, there were entities to classify correct.

Figure 6.10(b) shows that LRA is rather consistent across all concepts except *Computer Mouse*. LRA classifies almost every entity as correctly extracted, hence the recall for every concept is almost always close to one while the precision is almost always nearly 0.5.

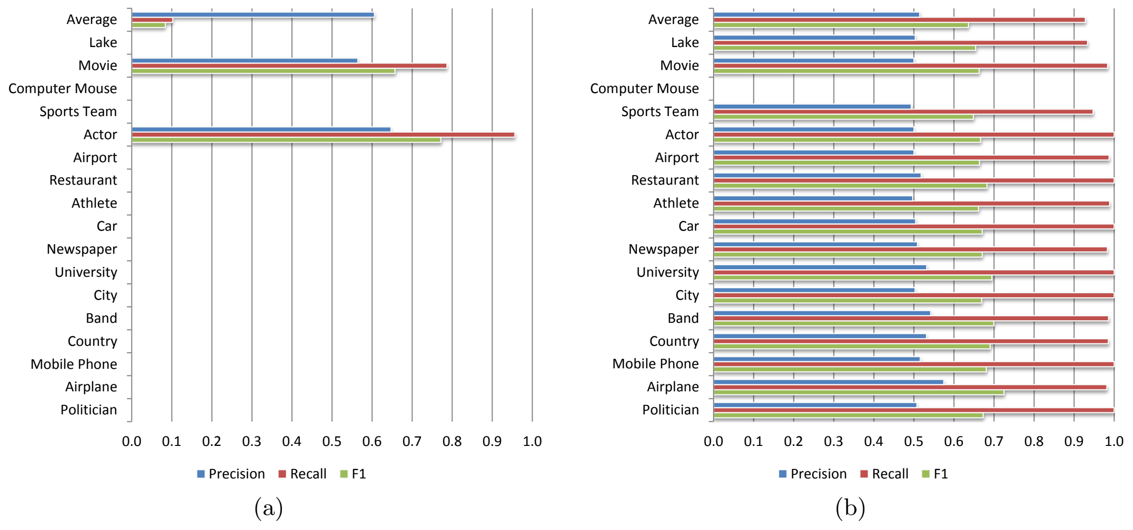


Figure 6.10: Performance of PMI (a) and LRA (b) per Concept

Figure 6.11(a) and Figure 6.11(b) show the performances of the text classification-based assessor and the Random Graph Walk-based assessor on a concept level using all the available training data. On average, the text classification approach yields almost equal precision and recall values. As expected, concepts in which entity names are more similar are classified more precisely. For example, the concept *Lake* often uses the word "Lake" in entity names and can therefore be classified more precisely by using the intrinsic n-gram features. The random graph walk assessor tends to either be selective, yielding a high precision (for example, for the concepts *Movie* and *Computer Mouse*) or it classifies all entities as correct, yielding a high recall (for example, for the concept *Lake*). Note that the precision for the concept *Band* is not zero, but rather undefined as no bands were classified to be correct.

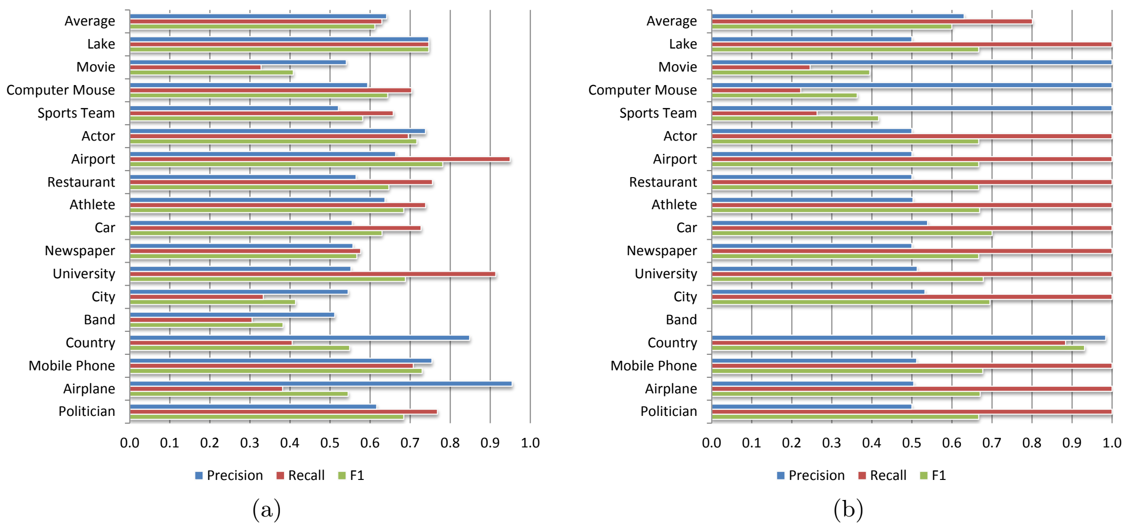


Figure 6.11: Performance of Text Classification (a) and RGW (b) per Concept

Figure 6.12(a) and Figure 6.12(b) show the performances of the naïve Bayes and the KNN assessors on a concept level. On average, both assessors are similar with a slight advantage to naïve Bayes, which benefits from more features than the KNN assessor. The only outlier is the concept *City* for which KNN outperforms naïve Bayes by almost 45 %. Cities might have similar features to countries, which can cause the confusion, but they are often syntactically more similar, which explains the KNN advantage. For example, “New” appears in multiple city names, and some cities even have the exact same name.

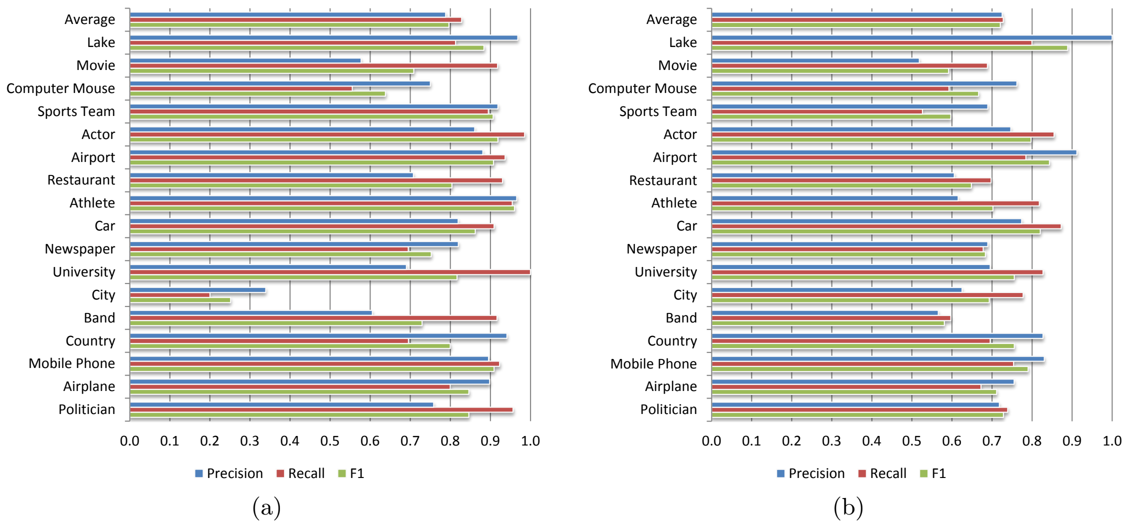


Figure 6.12: Performance of NB (a) and KNN (b) per Concept

Figure 6.13 depicts the classification performance of the combined assessor on a concept level. We can see fundamental differences between the combination and each of the single assessors. In general, the precision for each concept is much higher, while the recall is lower. The higher precision is due to the fact that both assessors have to independently classify an entity as correct. For the concept *Band*, for example, the assessors never agreed, hence no entity was considered correctly extracted. The precision for the concepts *Movie*, *Computer Mouse*, *Sports Team*, and *Country* increased to 100 % at the cost of a sharp drop in recall.

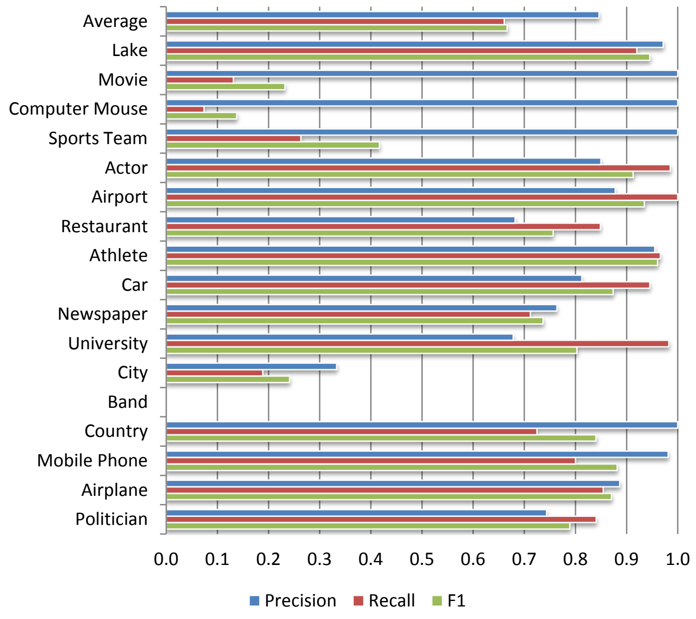


Figure 6.13: Performance of the Combined Assessor per Concept

6.3.3 Overall Comparison

Table 6.2 compares all evaluated assessors using 100 % of the training data without any trust thresholds. The naïve Bayes classification yields the highest F1 value and accuracy compared to all other techniques. LRA classifies almost everything as correct, yielding high recall at the cost of low precision. The combined assessor (RGW, NB, and Text) yields the highest precision (see Appendix B for a detailed threshold analysis of the five combined assessors).

The goal of the assessment algorithms is to filter out wrong extractions (high precision) while keeping correct extractions in the knowledge base (high recall). However, we put more importance on a knowledge base that contains fewer but usually correct extractions. To find out which assessor is better suited for the assessment of the entity extractions, we analyze the evaluation measures of the naïve Bayes approach and the combined assessor when we use a trust threshold.

6.3.4 Trust Threshold Analysis

As mentioned earlier, we prefer an accurate knowledge base to a knowledge base with many incorrect entities. We now analyze how to increase the assessment precision using trust scores from the assessors. The naïve Bayes assessor assigns trust scores to each entity classification indicating their confidence. These values are within the interval of $[0, 1]$. For the random

Assessor	**Precision**	**Recall**	**F1**	**Accuracy**
PMI	0.6059	0.1026	0.1754	0.5169
LRA	0.5144	**0.9280**	0.6620	0.5248
Text	0.6413	0.6291	0.6352	0.6235
RGW	0.6305	0.8009	0.7056	0.5587
NB	0.7881	0.8285	**0.8078**	**0.7966**
KNN	0.7254	0.7281	0.7268	0.7157
Combined (NB+KNN)	0.7931	0.8014	0.7972	0.7838
Combined (NB+Text)	0.7541	0.7553	0.7547	0.7507
Combined (NB+KNN+Text)	0.7925	0.7840	0.7882	0.7796
Combined (RGW+NB+Text)	**0.8694**	0.4363	0.5810	0.6788
Combined (RGW+NB)	0.8460	0.6610	0.7422	0.7577

Table 6.2: Overall Comparison of Entity Assessment Techniques

graph walk algorithm, we use the threshold parameter δ as shown in Equation 6.16 within the interval of $[0, 1]$ to increase ($\delta = 0$) or decrease ($\delta = 1$) the likelihood of classifying an entity as correctly extracted.

Figure 6.14 shows how the naïve Bayes approach performs when we increase the trust scores. The blue, vertical line at 0.5 shows the standard setting that we used to compare each assessment technique in Table 6.2. We can see that the precision increases only slightly between a trust threshold of 0.1 and 0.7. This behavior is due to the nature of the naïve Bayes classifier, which is prone to polarizing. "Polarizing" means that the classifier is always rather certain, which leads to very high scores for positively assigned instances (trust threshold > 0.7) and very low scores for negatively classified instances (trust threshold < 0.1). The highest precision we were able to achieve is about 85.6 % at a recall of about 45.1 %.

The recall keeps dropping above a trust threshold of 0.3, and because it drops quicker than the precision rises, the F1 value declines as well.

Figure 6.15 shows the trust threshold analysis of the combined assessor (RGW and NB). Unlike the naïve Bayes classifier, the precision scores for the combined assessor increase rather continuously between the trust threshold of 0.1 and 0.7. In general, the higher precision of the combined assessor compared to the precision of the naïve Bayes classifier comes at the cost of a lower recall. By raising the trust threshold to 0.8, we can reach a maximum precision of about 91.9 % at a recall of roughly 44.6 %.

Both assessors have their precision-recall break-even point at about 80 %, but we can call the combined assessor "better" in the sense that it can yield a higher precision than the naïve Bayes classifier, and reacts more predictably to an increased trust threshold. Figure 6.16 compares the two assessment techniques. Not only can we reach a precision gain of about 6 % using the combined assessor, the figure also shows that we reach the same precision as

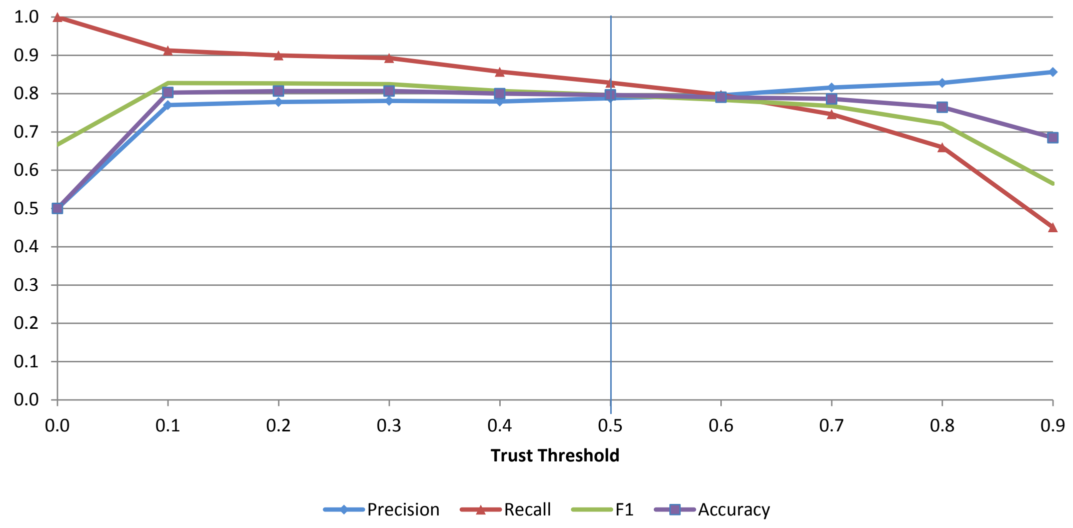


Figure 6.14: Trust Threshold Analysis of the Naïve Bayes Assessor

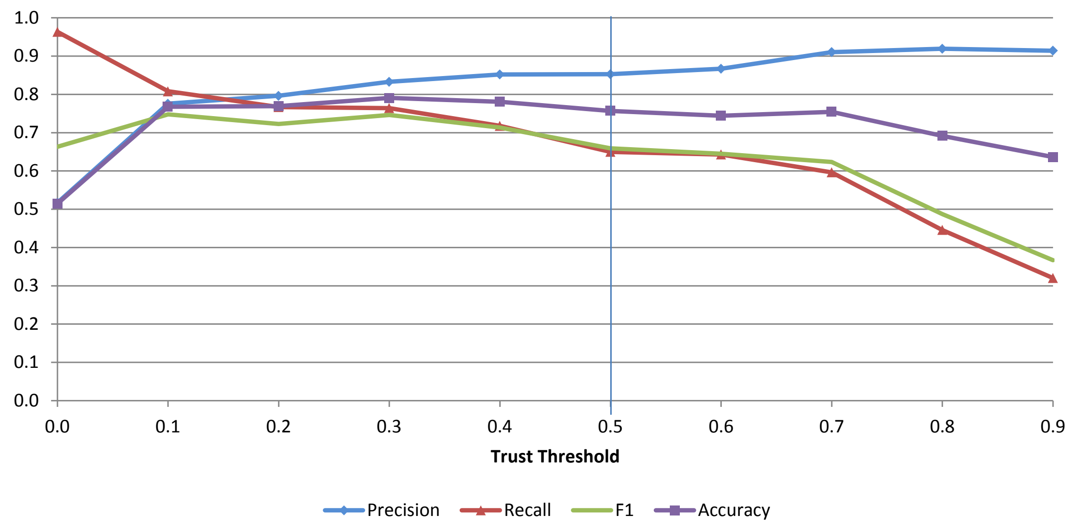


Figure 6.15: Trust Threshold Analysis of the Combined Assessor

the naïve Bayes classifier at a much lower trust threshold with a much higher recall. At a trust threshold of 0.4, the precision of the combined assessor is about 85 % at a recall of almost 72 %. The same precision can only be reached by the naïve Bayes classifier at a trust threshold of roughly 0.87 at a recall of only about 50 %.

6.4 Summary

In this section, we presented several assessment algorithms and evaluated them using extracted entities from our set of 17 concepts. The highest gain in precision of up to 41 % (from the baseline of 50 % to over 91 %, that is a relative gain of 82 %) was achieved using a combined assessor (random graph walk and naïve Bayes) at the cost of a recall drop of about 56 % (from

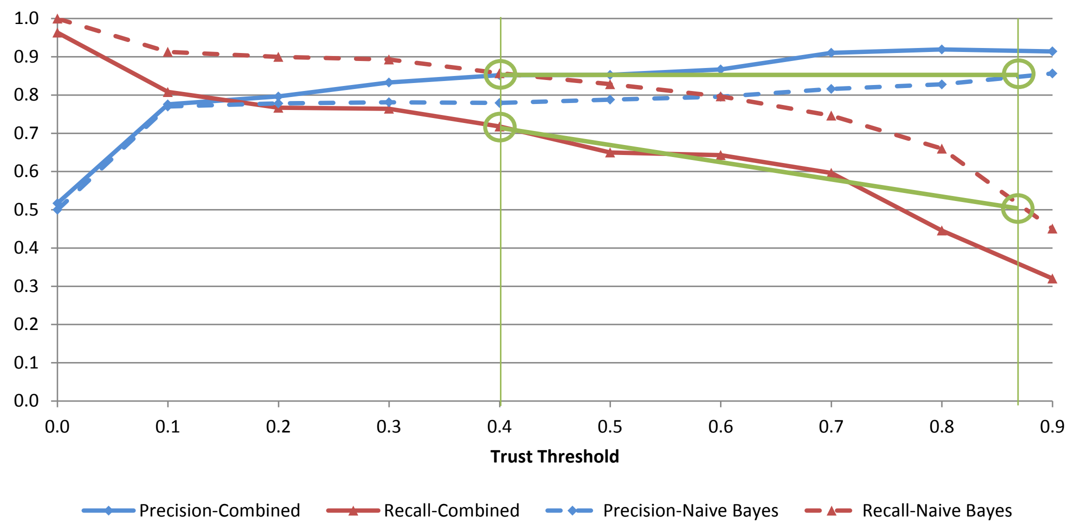


Figure 6.16: Comparison of the Trust Threshold Analysis between Naïve Bayes Assessor and Combined Assessor

100 % to about 44 %).

The precision-recall break-even point of the best assessors is at about 80 %, which is still a 30 % gain in precision at a loss of “only” 20 % recall. In Figure 5.21, we have seen the average precision values of the entity extraction techniques. The total average was about 45 %. Using the entity assessment techniques presented in this chapter, we can increase the average precision to over 80 % on average which meets our requirement that at least 80 % of the entities in the knowledge base are correct.

We confirmed Thesis 3 in this chapter, that is, we have shown that using small sets of training instances and a combination of machine learning algorithms can outperform current state-of-the-art algorithms for entity assessment such as PMI.

Chapter 7

Extraction of Facts

This chapter introduces the design of a component for extracting factual information from the Web. Until now, we have shown how we can extract entities for certain concepts from the Web. These concepts have attributes attached, and this chapter explores the extraction of facts for these attributes. First, the knowledge to be extracted is encoded in an ontology. Then, entities are given and the fact extraction process completely automatically finds the values for the specified attributes. We use an assessment phase to assign trust values to the extractions before storing the found facts. This approach enables us to filter facts for which the extraction system is low in confidence.

Figure 7.1 depicts the fact extraction cycle, which runs in an infinite loop until the time slot for fact extraction expires and the next extraction component takes over. First, the ontology is loaded. All concepts, and all entities for each concept, are then examined and follow the extraction pipeline. For each concept and entity, all queries for all attributes are generated and result pages from search engines and Semantic Web indices are retrieved. On these result pages, the five extraction techniques that examine different structures and formats are processed. After processing, the system calculates and updates the trust value for all extractions in the cycle and stores the results. The following sections describe the ontology engineering process, the retrieval of result pages, the five extraction techniques, and the trust calculation. The last section of this chapter evaluates the described approaches in detail.

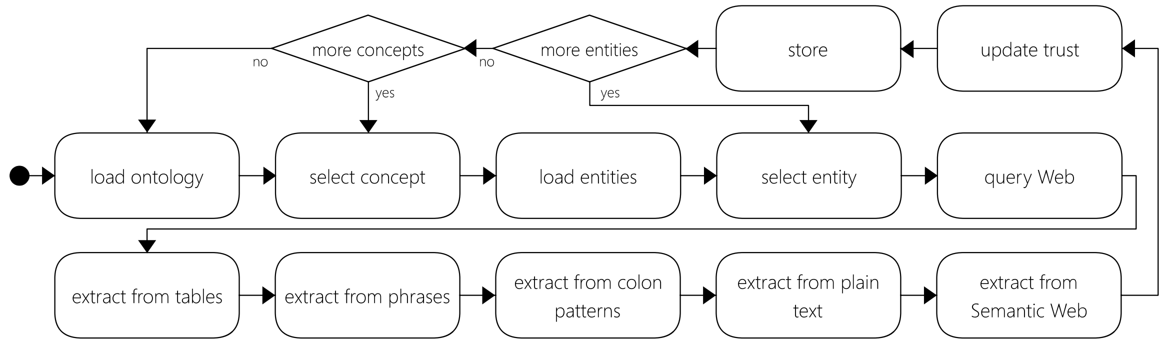


Figure 7.1: Overview of the Processes in the Fact Extraction Cycle

7.1 Ontology Engineering

Before the extraction process can begin, WebKnox needs to know which concepts, attributes, and entities exist. This knowledge is called prior knowledge. The prior knowledge for WebKnox is modeled in an ontology using the Web Ontology Language (OWL) (Schreiber, 2004). Therefore, all concepts and attributes are defined in the knowledge ontology and the entities and facts that are extracted are stored in another separate data ontology.

The process of creating an ontology is often called ontology engineering. This process requires a domain expert to create and refine an ontology. The fact extraction process of WebKnox requires the names of attributes and concepts to be the same as they are found on the Web. Intuitively, a domain expert would visit Web pages that display relevant information about an entity of a concept and copy the attribute names into the ontology. This manual process is very labor intensive. For this reason, we created a tool called "Ontofly" (Willner, 2010, Urbansky et al., 2010, Willner, 2011) to ease this process. The tool is explained in more detail in Section 10.1. In this section, we will only describe the conceptional ideas used in the ontology engineering process.

7.1.1 Datatypes

The purpose of the knowledge ontology is to define the knowledge represented in the data ontology and to serve as an input for the extraction process. An OWL datatype property is assigned to every attribute in the knowledge ontology. This property determines which type of value the attribute will have. It can also be used by other programs reading the ontology, that try to parse the data, and to guide the extraction process in finding matching extraction candidates for the attribute. A datatype property can have any XSD datatype[1], but WebKnox uses only the following:

1. **String**: A string is a sequence of characters. WebKnox, however, only considers proper nouns as fact candidates for a string attribute, that is, only a sequence of words starting with a capitalized character or a number are considered to be possible answers.
2. **Boolean**: Attributes with a boolean value have either true or false as a value. WebKnox only searches boolean values in tables and looks for "yes" and "no" occurrences.
3. **Decimal, Double, Float, Integer, Int, Long**: WebKnox handles these numeric attributes identically. Every numeric attribute is treated as a double and only numbers are extracted as fact candidates for the attribute.
4. **Date**: An XSD date is a string given in a standardized UTC format: YYYY-MM-DD. WebKnox looks for several representations of dates from Web sources and tries to transform these back to the UTC format.
5. **AnyURI**: The Uniform Resource Identifier is a string in a special format pointing to a resource. All correctly formatted URIs are taken as candidates by WebKnox for attributes with this data type.

[1]`http://www.w3.org/TR/xmlschema-2/\#d0e11239`, last accessed on 4th of February 2012

6. **AnyType**: Attributes with values that do not match any previously mentioned datatype can have the AnyType property. WebKnox takes all characters around or after the attribute on the Web source into account when determining the fact candidates. Thus, AnyType can be used for strings that are not proper nouns.

Each of these datatypes is internally mapped to a regular expression. The mapping is shown in Appendix D in Table D.1.

Many attributes have special formats, which would need to be set to the AnyType datatype. This datatype, however, gives WebKnox no information about how the expected value is formatted. For example, the *geographic coordinates* attribute for the *Country* concept has a fixed format with numbers and letters, such as *27 00 S, 133 00 E.* Using the AnyType datatype, WebKnox would not know the appropriate format and would extract not only the correct value, but also incorrect text before and after this value. To solve this problem and make the approach more practical, we include another annotation called "regularExpression" in the OWL definition of the datatype object. If such an annotation exists, WebKnox will use the expression and overwrite the given datatype.

7.1.2 Unit Types

Knowing the unit type of the expected answer benefits the fact extraction process for numeric attributes. The *length* of a *River* should be a physical length (in meters, for instance) while the *length* of a *Movie* should be a time (in minutes or seconds, for instance). Knowing the expected measurement for numeric attributes aids the extraction process by ensuring that numbers in the appropriate units (for example, "meters" or "km" for the concept *River*) obtain a higher confidence than values without a unit.

When extracting numeric facts from the Semantic Web, we even need to know the exact unit that belongs to the facts for each predicate. For example, two triples from different ontologies might appear as follows:

```
<dbpedia:Nile> <dbpedia:length> "6650000"
<freebase:Nile> <freebase:length> "6650"
```

DBpedia normalizes the length values to meters, while Freebase normalizes to kilometers[2]. Not knowing the unit of a number would lead us to two different results, although they should corroborate each other since they resolve to the same value.

7.1.3 Ranges

Since the main purpose of the ontology is to guide the extraction process, we introduce a few new vocabulary terms to specify the range of values for an attribute. These terms use the WebKnox namespace "wx:" (the URI of the namespace is `http://webknox.com`).

[2]This is not really the case, but it serves as an example to explain the problem.

1. The *hasRange* property connects an ontology attribute with a blank node. More information about the range is attached to this blank node. One attribute can have several *hasRange* properties.
2. The *rangeType* is the type of range we want to specify. We distinguish between “minmax” and “possible”. The term “minmax” means that we know the numeric range of values for the attribute, while the term “possible” means that we know all possible values that the attribute can have.
3. The properties *rangeMin*, *rangeMax*, and *rangePossible* specify the values in the range of the attribute. If the rangeType is “minmax”, we specify the minimum and maximum values with “rangeMin” and “rangeMax” respectively. If the rangeType is “possible”, we enumerate all possible values using “rangePossible”.

Figure 7.2 shows an example graph for the attribute *memory*. We see that the attribute can belong to several domains (*Phone* and *Camera*), which we declare using rdfs:domain. We use rdfs:range to specify that the values are integer values and make it an OWL “datatypeProperty” using rdfs:type. We now declare one range for the *memory* attribute using wx:hasRange and connect it with a blank node (rn1). We use rdfs:domain again to declare that the blank node specifies the range type for the *memory* attribute that belongs to the *Phone* concept since the *memory* attribute also belongs to the *Camera* concept but might have different ranges in that concept. Finally, we use wx:rangeType, wx:rangeMin, and wx:rangeMax to declare that the *memory* attribute for mobile phones always has a minimum value of 0.5 megabytes and a maximum value of 32,000 megabytes. The extraction process can now use this information and discard all extractions that do not fall into the defined range.

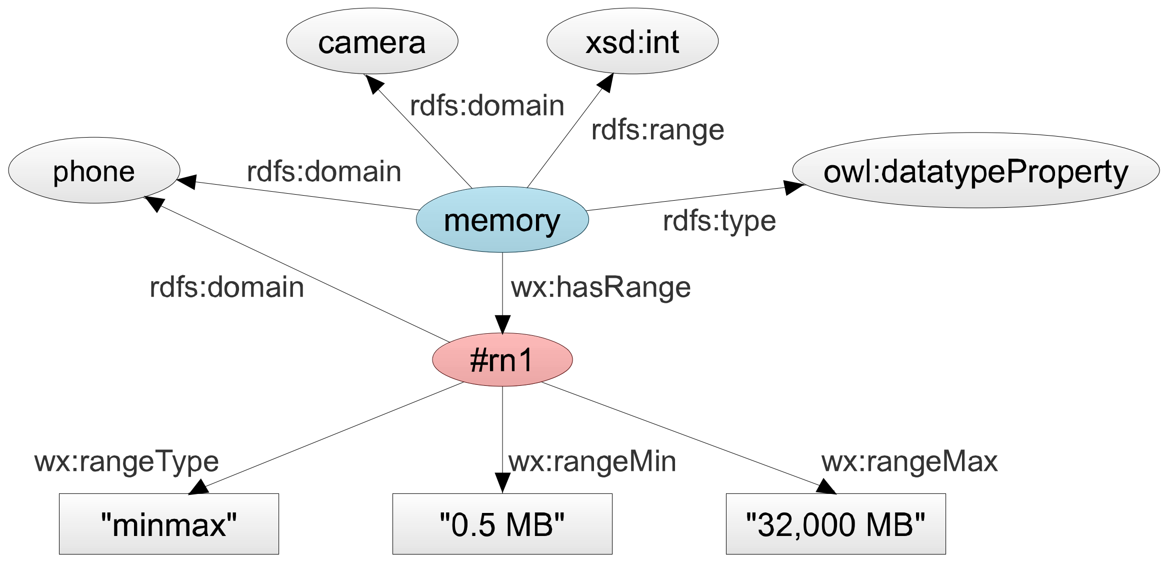


Figure 7.2: Modeling Range Types Using Blank Nodes

7.2 Related Work

In this section, we review related work in the field of fact extraction. First, we review complete systems that are in some aspects comparable to WebKnox, and second, we review assessment algorithms that can be used to assign confidence scores to the extracted facts.

7.2.1 Related Systems

KnowItAll (Etzioni et al., 2004) is an unsupervised system that can automatically extract facts about entities from the Web. KnowItAll is redundancy-based, which means it relies on the assumption that a fact occurs many times on the Web. The system mainly extracts facts from plain text, but much information, especially numbers (for example, the *population* of a *Country*), is given in table structures that are not evaluated by KnowItAll. Also, the PMI score for validating the extractions would most likely not work with numeric extractions as the redundancy assumption does often not hold for specific numeric values. We differ from KnowItAll in that we make use of HTML structures to extract fact mentions precisely.

Textrunner (Yates et al., 2007) is a system that builds upon the concepts of KnowItAll but without requiring user input. Textrunner extracts information from a corpus of Web pages in three steps: (1) The noun phrases of the sentence are tagged, (2) nouns that are not too far away from each other are put into a candidate tuple set, and (3) the tuples are analyzed and classified as true or false. Our approach is very different from Textrunner. Textrunner avoids ontologies allowing for “open information extraction” at the cost of precision. We use an ontology which allows us to guide the extraction process and obtain more precise results. Both approaches have their advantages and disadvantages. Our disadvantage is that we need initial labor to create the ontology, but the advantage is the quality of the extractions. Textrunner does not “know” which entities, concepts, and attributes belong together. It is therefore difficult or even impossible to find the *display size* of a phone, for example, if Textrunner does not find it mentioned in some text within one of the rigid rules it uses[3]. TextRunner also focuses more on relations between concepts and entities than on entities and their facts.

GRAZER (Zhao and Betz, 2007) is a system that learns new facts from the Web. The input for GRAZER is seed facts (attribute-value pairs) for given entities. Entities and seed facts are automatically generated using specialized wrappers. The system then obtains relevant pages for the given entities. Relevant pages are those that mention the entity in the text. On these pages, the seed facts are corroborated and new facts are extracted. The system searches for mentions of the seed facts on the relevant pages and adds the source if the fact is found on the page. The corroboration happens in free text and in structured HTML as all tags are removed and only the area around the attribute name is searched for the mention of the value. Although GRAZER does not need an ontology of the knowledge domain as an input it relies on a set of seeds for entities and facts. These seeds are obtained in a non-generic way by inputting the data by hand, which is labor intensive or by scraping sources with specialized wrappers. The same facts are extracted several times and are treated as new facts when they have a different attribute that is just a synonym, for example, “Birthday: 17.01.1962” is

[3]The system can be tested on `http://openie.cs.washington.edu/`, last accessed on 4th of February 2012.

different from “Date of Birth: 17.01.1962”. Similar to Textrunner, GRAZER does not use an ontology and therefore has little information about the type of relations it can expect between entities and facts. For example, it is difficult to transform the extracted factual information to Semantic Web RDF triples in a normalized manner since the datatype (XSD) is unknown and the normalization has not taken place.

DBpedia (Auer et al., 2007) is the Semantic Web equivalent of Wikipedia. Most of the valuable information in DBpedia comes from the so-called “Infoboxes” from Wikipedia. Auer et al. (2007) created certain extraction, normalization, and mapping rules to store the data in a machine-readable manner as RDF triples. The drawback of DBpedia is that any information not contained in Infoboxes and, more importantly, not in Wikipedia, will not make it to the RDF store. We differentiate from this approach in that we do not rely on a single Web Source and hard-coded wrappers for certain tables, but rather aim to extract information from arbitrary Web pages.

7.2.2 Fact Assessment

Traditional information extraction focuses on extracting as much information as possible from a small corpus whereas Web information extraction systems often rely on the redundancy of Web content (Yates, 2007). This corpus shift means that the focus of the extraction techniques should be set on precision since a high recall can be achieved by relying on the high number of mentions of the entity or fact that is extracted. One major problem with Web information extraction systems is that the quality of the extractions can vary, for example, extracted entities may not really belong to the concept they were assigned to or extracted facts are wrong. In Chapter 6, we discussed several methods to assess extracted information. For extracting factual information, the following two approaches are mentioned in literature frequently:

1. **Simple Scoring**: For the fact extraction task a simple scoring based on the number and quality of sources can be used to decide which fact extraction is correct and which is not (Zhao and Betz, 2007). The effectiveness of simple scoring relies, however, on the assumption that correct facts are extracted more often than incorrect facts which also depends on the extraction technique and the type of the fact. Rare facts, for example, might be extracted correctly but do not score very high.
2. **Pointwise Mutual Information**: Etzioni et al. (2004) use patterns as discriminators to ensure the correctness of an extracted fact or entity. These discriminators are transformed to queries for Web search engines and the PMI between the extraction and the discriminator using the hit counts is computed (see also Section 6.1.4).

7.3 Retrieving Fact Pages

Retrieving relevant pages that contain facts is a crucial process that has to be tightly coupled with the extraction process. The source retrieval process uses the names of the searched for entities and attributes as input data. The process then queries a search engine and outputs the retrieved pages together with information about which attributes are expected on the page. This output is then fed into the extraction process. The focus lies on retrieving semi-structured HTML pages as they are easy to access via generic search engines such as Google and are the predominant format on the Web[4]. However, we also search the Semantic Web of Linked Data to find mentions of facts.

To retrieve sources that contain the searched facts, WebKnox uses two kinds of generic queries:

1. **Multi-attribute queries**: These kinds of queries try to find pages relevant to the entity and extract all searched facts from the retrieved pages (for example, the query "Australia"). It is expected that the highest ranked pages the search engine returns are information rich and show the entity from different angles. Therefore, we search for all attributes connected to the concept to which the entity belongs. For querying the Semantic Web we only use this type of query.
2. **Single-attribute queries**: These kinds of queries focus on a single attribute at a time (for example, "Australia population"). Using these very precisely targeted queries allows us to find more relevant pages that definitely mention the sought after information piece.

The retrieved sources are then passed to the fact extraction process along with information about the type of the query. Using this information, the system only extracts single attributes on pages returned from single-attribute queries, but tries to find all attributes on pages retrieved from multi-attribute queries.

7.4 Extraction Techniques

After WebKnox retrieves the sources, it uses different techniques to extract the facts for which it knows it should search. We can increase the quality of extracted facts by using different extraction structures for different types of facts. As covered in the background, a common approach for fact extraction is to use the complete Web page content and simply remove all HTML tags (as done by the GRAZER system (Zhao and Betz, 2007)). This method, however, also removes all advantages that come with the semi-structured type of HTML documents. WebKnox differs from current approaches because it takes the different generic formats and structures into account that are used to represent facts on Web pages. Furthermore, WebKnox also searches and extracts fact mentions from the Web of Data.

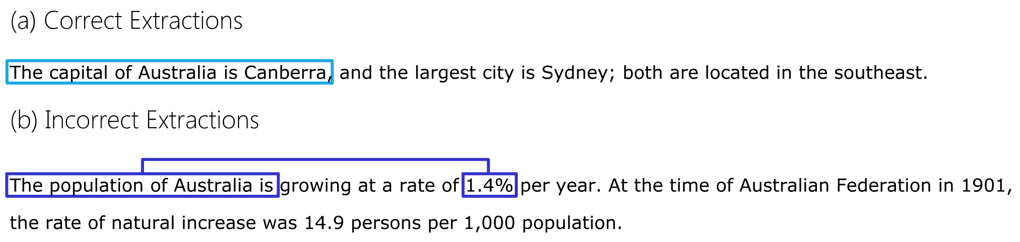
WebKnox employs five extraction techniques that focus on the different structures in which the facts can be found. These techniques are Phrase Extraction, Table Extraction, Colon Pattern Extraction, Plain Text Extraction, and Semantic Web Extraction.

7.4.1 Phrase Extraction

Phrases are natural language representations of facts for a specific entity. For example, Figure 7.3 (blue box in (a)) shows an example of how the phrase "The capital of Australia

[4] `http://www.google.com/help/faq_filetypes.html`, last accessed on 25th of March 2012

is Canberra" is used on a website. The phrase covers the fact "capital:Canberra" for the entity *Australia*. Part (b) in Figure 7.3 shows, however, that these patterns can also lead to incorrect extractions (purple boxes). There are only a small number of generic phrases that can be applied to many different concepts and attributes, but these often lead to very reliable results. This occurs because, ideally, the searched value for the attribute appears right after "is" in the phrase. WebKnox uses only two phrases: "the `ATTRIBUTE` of `ENTITY` is" and "`ENTITY`'s `ATTRIBUTE` is". The source retrieval process also uses these phrases to discover pages containing them.


(a) Correct Extractions

The capital of Australia is Canberra, and the largest city is Sydney; both are located in the southeast.

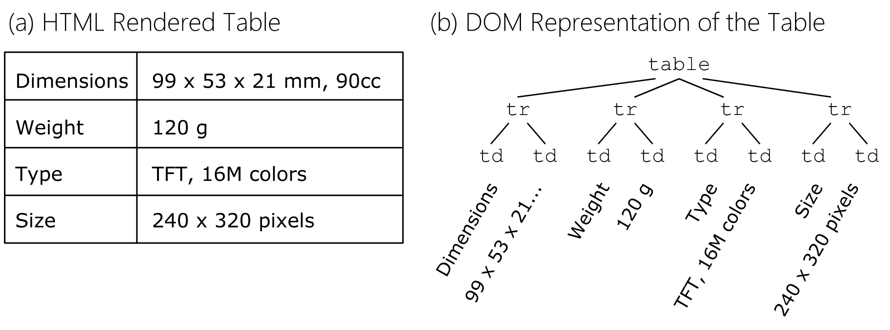
(b) Incorrect Extractions

The population of Australia is growing at a rate of 1.4% per year. At the time of Australian Federation in 1901, the rate of natural increase was 14.9 persons per 1,000 population.

Figure 7.3: Example of Correctly (a) and Incorrectly (b) Extracted Facts from Phrases

7.4.2 Table Extraction

Tables are important HTML structures on the Web used to represent factual information. Their prevalence has led to numerous wrapping techniques. Maintaining the HTML structure allows us to traverse in the DOM tree of the Web page and find corresponding attribute-value pairs in tables. Figure 7.4(a) provides an example of a rendered HTML table and Figure 7.4(b) exhibits the DOM representation of the table. This is a simple example of a table, but also a very common one. By identifying the `td`-tag with the attribute, the sibling `td`-tag with the value can be found and only the text inside that element is extracted. Extracting a fact from a table is often more reliable than from free text because the boundary for the value is given by the `td`-tags.

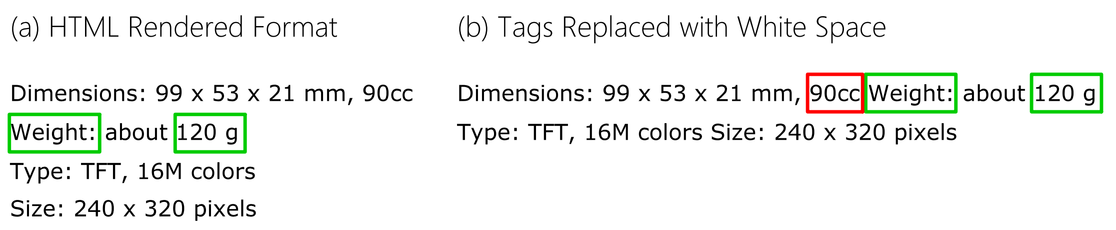

(a) HTML Rendered Table

Dimensions	99 x 53 x 21 mm, 90cc
Weight	120 g
Type	TFT, 16M colors
Size	240 x 320 pixels

Figure 7.4: Example Table for Mobile Phone Specifications

7.4.3 Colon Pattern Extraction

Colon pattern text refers to the text directly following a colon (`:`). Facts are often given in an unstructured way (no HTML tags), but with the format `ATTRIBUTE:VALUE` so that only the text after the colon needs to be extracted. Figure 7.5 shows an example of this representation. Figure 7.5(a) depicts the HTML rendered version while Figure 7.5(b) shows the text as it is seen when the separating tags are removed (replaced with whitespaces). If one would try to extract the weight attribute (*weight* in Figure 7.5(b)), expecting a numeric value and not recognizing the format, the *90 cc* would be extracted as it is closer to the weight attribute than the correct value *120 g* after the colon. The colon pattern can therefore increase the fact extracting precision in a very simple manner.


(a) HTML Rendered Format

Dimensions: 99 x 53 x 21 mm, 90cc
Weight: about 120 g
Type: TFT, 16M colors
Size: 240 x 320 pixels

(b) Tags Replaced with White Space

Dimensions: 99 x 53 x 21 mm, 90cc Weight: about 120 g
Type: TFT, 16M colors Size: 240 x 320 pixels

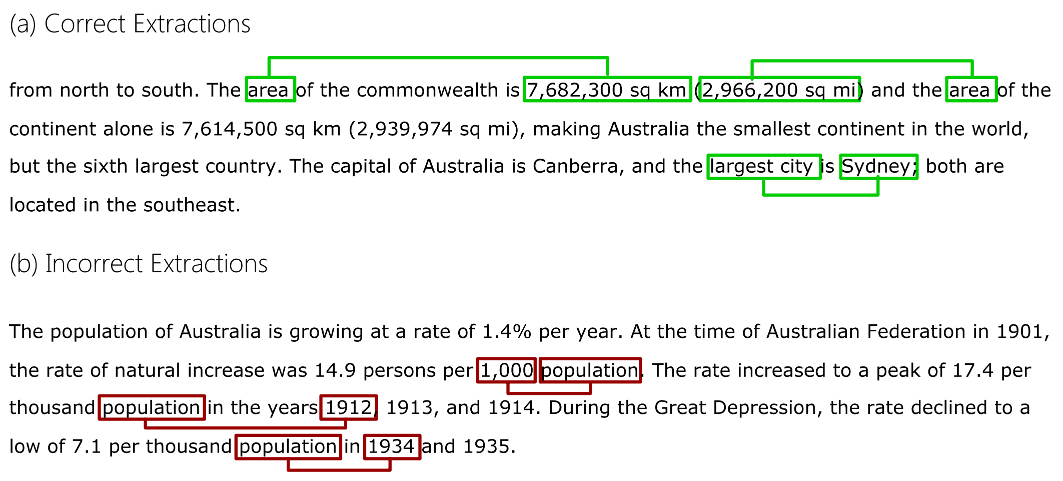
Figure 7.5: Example of Fact Representation in a Colon Pattern, with HTML Structure (a) and without HTML Structure (b)

7.4.4 Plain Text Extraction

Plain text is the absence of structure (tags) and additional formatting (phrase or colon pattern). Facts can also appear in long paragraphs of text, but since no further information about the structure and format is given, all text around the attribute must be considered a valid answer for the attribute's value. It is always assumed that the next matching value closest to the attribute is extracted. WebKnox takes the sentence in which the attribute appears as the boundary. This way incorrect information further away is not extracted as well. Figure 7.6(a) provides an example of free text that states facts about *area* and *largest city* of the entity *Australia*. The connected green boxes represent correct attribute-value pairs found in the text. However, information found in free text is the least reliable, which can be seen in Figure 7.6(b) in which the red boxes represent incorrect extractions for the attribute *population*. Nevertheless, using information found in free text increases the recall and should still be considered, especially for rare facts that do not appear in tables or other structures and formats.

7.4.5 Semantic Web Extraction

The Semantic Web contains billions of statements in the form of triples. Many of these statements are facts about entities. Hence, the Semantic Web is a valuable source for factual extraction and corroboration of already extracted facts. While the previously described techniques focus on the extraction of facts from the Visible Web where we deal with semi-structured HTML pages, the Semantic Web Extraction technique retrieves RDF triples.



(a) Correct Extractions

from north to south. The area of the commonwealth is 7,682,300 sq km (2,966,200 sq mi) and the area of the continent alone is 7,614,500 sq km (2,939,974 sq mi), making Australia the smallest continent in the world, but the sixth largest country. The capital of Australia is Canberra, and the largest city is Sydney; both are located in the southeast.

(b) Incorrect Extractions

The population of Australia is growing at a rate of 1.4% per year. At the time of Australian Federation in 1901, the rate of natural increase was 14.9 persons per 1,000 population. The rate increased to a peak of 17.4 per thousand population in the years 1912, 1913, and 1914. During the Great Depression, the rate declined to a low of 7.1 per thousand population in 1934 and 1935.

Figure 7.6: Example of Correctly (a) and Incorrectly (b) Extracted Facts from Plain Text

In order to extract information from the Semantic Web we need to map attributes from the ontology to predicates in the Semantic Web. We could employ a fuzzy matching technique using the attribute's name, but since URIs do not have to be human readable and might be cryptic, we decided to opt for the direct mapping approach.

Additionally, as mentioned in Section 7.1.2, we need to specify the unit belonging to numeric predicates. A complete mapping of an attribute of the WebKnox ontology to a predicate and a unit type is shown in Table 7.1.

WebKnox Attribute Name	**Semantic Web URI**	**Unit**
length	http://dbpedia.org/ontology/length	"m"
weight	http://dbpedia.org/ontology/weight	"g"

Table 7.1: Example of an Attribute Mapping to a Predicate and Unit

The extraction works in the following steps:

1. We query an index of the Semantic Web[5] with the entity name alone.
2. In the result list, we get all URIs that have the entity name as a label.
3. For each candidate URI, we then get all triples and add all URIs that are in an OWL "#sameAs" relationship with the candidate URI. This step allows us to find facts present in other ontologies about the exact same entity.
4. We get all triples for all URIs in the extended candidate list. For each triple, we try to find the mapped entity (see Table 7.1) for the given predicate in the triple. If a mapping is found, we extract the value. If the attribute is a numeric attribute, we normalize the value with the given unit information.

[5]The best freely available index at the time of this writing (4th of February 2012) is `http://sindice.com/`.

Intuitively, if we find a fact using several techniques, it is likely correct. We will show the value of the different techniques in the evaluation, for preliminary results see also Urbansky (2009). The next section describes how the trust in extracted facts is calculated.

7.5 Assessing Fact Extractions

Once we have extracted facts, we rank them in order to determine the fact value that is most likely to be the correct value for the attribute. It is now necessary to find the correct values by assigning **trust** to each extraction.

The **absolute trust** is a non-negative number that indicates the confidence in the extraction. The higher the number, the more reliable the extracted value.

The following equations assign trust values and aim to improve the ranking of the extracted values.

The easiest way to rank the extracted values is to count the number of extractions. The more often a value has been extracted, the higher the trust value. Equation 7.1 shows how the trust value is calculated in this case, with N being the number of extractions for the given value, and x being a tuple consisting of concept, entity, attribute, and value: $x = \langle x_{concept}, x_{entity}, x_{attribute}, x_{value}\rangle$. This way of assigning a trust value is called *QuantityTrust* from now on.

$$QuantityTrust(x) = N \tag{7.1}$$

The Quantity Trust does not make use of additional information such as **where** (the source) and **how** (extraction technique) the fact was extracted. This information should, however, be considered when determining the trust for an extraction.

7.5.1 Determining the Source Trust

Some pages that are retrieved for the extraction process mention the attribute and its value several times. For example, suppose a page is retrieved when searching for the entity *Nokia N95* and the attribute *talk time*. The page mentions the attribute several times, two times with the correct value of 6.5 hours, but three times with different values that do not relate to the entity but to other mobile phones. The source trust can therefore be reduced whenever there is more than one value for the searched attribute, as shown in Equation 7.2 where D is the number of different values found for the given attribute and source. The source trust can have values between zero and one with one being the most trust and zero being no trust.

$$SourceTrust(attribute, source) = \frac{1}{D} \tag{7.2}$$

7.5.2 Determining the Extraction Technique Trust

Extraction techniques have different levels of precision, which must be taken into consideration when calculating the trust for a fact. The values determined in the test set are not representative of all possible concepts and domains. Since WebKnox aims to be domain independent, the precision values determined for the test set cannot be taken as references. WebKnox uses self-supervised machine learning to automatically estimate the trust for the five extraction techniques used (Urbansky, 2009). The trust value for the extraction techniques is an estimated precision, meaning it is a number between zero and one with one being highest trust (all extractions were correct) and zero being no trust (all extractions were incorrect).

For all extraction techniques e, information about the number of extractions $N(e)$, and the number of correct extractions $C(e)$ is kept. The $ExtractionTechniqueTrust$ is then calculated as the ratio of correct extractions to total extractions (Equation 7.3):

$$\mathit{ExtractionTechniqueTrust}(e) = \frac{C(e)}{N(e)} \tag{7.3}$$

Initially all extraction techniques are initialized with a trust value of 0.5. The next three steps are then as follows:

1. The input for the first step is the extraction result with an assigned trust. In the first step, the most trusted fact is searched throughout all concepts and attributes. It is then assumed that this fact is really correct, since it has a high trust. All extraction techniques used to extract that very fact value get credit for a correct extraction, that is, $C'(e) = C(e)+1$ and $N'(e) = N(e)+1$. Extraction techniques that led to wrong fact values for that attribute, are penalized for a wrong extraction, that is, $N'(e) = N(e)+1$. In the next iteration, this highly trusted fact is no longer considered when looking for the highest trust.

2. In the second step, the trust for the extraction techniques is updated based on the number of correct and total extractions that have been revised in the former step, that is, the *ExtractionTechniqueTrust* is recalculated using Equation 7.3.

3. In the third step, the trust for all extracted values is recalculated using the updated trust for the extraction techniques. After this step, the ranking of the extracted values for each attribute could change. The newly-ranked list is then input for the first step to repeat the process. The iteration can be stopped when the trusts for the different extraction techniques converge, that is, when the trust values no longer change considerably. In case the trusts never converge, the iteration will only stop after all highest trusted facts have been evaluated in step one.

7.5.3 Combining Source and Extraction Technique Trust

Taking both the source trust and the trust in the extraction technique into consideration, the trust for an extracted value can be calculated as shown in Equation 7.4. S is the set of sources from which the given fact has been extracted, $ExtractionTechniqueTrust(e)$ is the trust of the extraction technique e used, and $SourceTrust(s)$ is the trust of the s^{th} source. The trust will therefore be high when the value has been extracted in many trustworthy sources using numerous highly trusted extraction techniques. This trust formula shall be called *CombinedTrust*.

$$CombinedTrust'(x) = \sum_{s \in S}(\sum_{e \in E} ExtractionTechniqueTrust(e) \times SourceTrust(x_{attribute}, s)) \tag{7.4}$$

7.5.4 Normalization

Facts can be presented in different formats yet still represent the same information. For example, dates can be written in many ways, such as *January 17th, 1962* or *17/01/1962*. Moreover, many numeric facts have units. Not taking the unit into account leads to the extraction of two unique facts when actually only a single fact is mentioned; for example, *2 inch* and *5.08 cm* is the same fact presented in different units. Normalization helps locate facts in different formats and clusters them.

7.5.5 Validating Numeric Fact Values across Entities

Another problem with extracted facts is that some attributes do not have a single absolutely correct value. The *population* attribute, for instance, is not mentioned correctly on any website on the entire Web as it changes almost every second[6]. Instead, there are values that are almost the same and can be considered correct. Fact values for attributes with fuzzy values tend to not corroborate well. Consider the following fact values that might have been extracted for the *population* attribute for the entity *Australia*:

```
300 (3 times)
21,000,000 (1 time)
21,340,000 (1 time)
22,578,420 (1 time)
20,452,340 (1 time)
```

The problem here is that the exact same number for the population is not mentioned on more than one source. The incorrect extraction *300*, however, is extracted several times and therefore gains higher trust.

To solve this problem, two assumptions are made:

1. The order of magnitude (OOM) for numeric facts is often the same for entities within the same concept. Exceptions to this include population of countries and prices for products.

[6]Example for a constantly changing fact value: `http://www.usatoday.com/news/nation/2006-10-12-population-milestone_x.htm`, last accessed on 25th of March 2012

2. There are well-known entities that appear on many Web pages. Due to the high number of mentions, values for numeric attributes can be extracted with a relatively high trust.

Both assumptions were supported in our test set for most of the fact values. A larger test set with more (and more well-known) entities would most likely further support this assumption.

To take advantage of the fact that the OOM is often the same, WebKnox uses a validation process across all entities for a given attribute. This process is called *CrossValidation* and is part of the second step in the self-supervised learning loop. It works as follows:

1. For all numeric attributes, an OOM distribution is constructed.
2. If the highest trusted value from the first step of the learning loop is a numeric value, the number is considered to be correct and the OOM of that number is given credit in the attribute's OOM distribution.
3. In the next iteration, the trust in the fact values of the same attribute will be calculated as shown in Equation 7.6. The *CrossValidationFactor* for a numeric fact value is one plus the support of the OOM, which is a number between zero and one with one being 100 % support (all other entities of the concept had values with exactly the same OOM for that attribute) and zero being 0 % support (no other entity of the same concept had the same OOM for that attribute).

$$\mathit{CrossValidationFactor}(x) = \mathit{1} + \mathit{support}(\lfloor \mathit{log}_{\mathit{10}}(x_{value}) \rfloor, x_{concept}) \tag{7.5}$$

$$\mathit{CombinedTrust}(x) = \mathit{CrossValidationFactor}(x) \times \mathit{CombinedTrust}'(x) \tag{7.6}$$

The cross validation component has been evaluated in Urbansky et al. (2008) and Urbansky (2009).

7.6 Evaluation

In this section, we evaluate the performance of the fact extraction techniques on a test ontology. First, we describe how we created the ontology and what we consider the ground truth. Then, we evaluate the extraction techniques, how well the extraction works for different datatypes, how well it works for different concepts, and how we can increase the precision by using the extraction trust. See also Urbansky et al. (2008) and Urbansky (2009) for a comparison of four of the described extraction techniques with the GRAZER extraction system.

7.6.1 Ontology

Figure 5.17 shows the concepts of the test ontology for WebKnox that have already been used in the evaluation of the entity extraction. For each of the 17 concepts we selected 10 entities. Whenever possible, we selected the entities for each concept randomly; for some concepts we searched for lists of well-known and lesser-known entities and took every n^{th} entity where n is the length of the list divided by ten. This way we have a range of popular and less-popular entities for each concept in the evaluation ontology. The Table C.1 in Appendix C lists all 170 entities used for the evaluation.

Table 7.2 shows the complete ontology, including the attributes for each concept, the number of synonyms for these attributes, the number of mappings of the attributes to Semantic Web URIs, the attributes' datatypes, and the expected units of the attributes. In total, we assigned 101 attributes (1,010 facts in total) to the 17 concepts. These 101 attributes divide into the five datatypes as follows: 36 String, 43 Numeric, 8 Date, 6 AnyURI, and 8 Boolean. The attributes that have a datatype with a * symbol in Table 7.2 have special regular expressions attached to them. For example, *dimensions* are usually given in length x width x height, (for example, *3cm x 5cm x 10cm*), which can easily be captured with a regular expression.

Concept	**Attribute**	**#Synonyms**	**#Mappings**	**Datatype**	**Unit**
Actor	Birth Name	1	1	String	-
	Place of Birth	3	3	String	-
	Date of Birth	2	2	Date	-
	Height	0	3	Numeric	cm
Athlete	College	0	4	String	-
	Place of Birth	3	3	String	-
	Nationality	0	2	String	-
	Date of Birth	2	2	Date	-
	Height	0	3	Numeric	cm
Airplane	Manufacturer	0	2	String	-
	First Flight	0	3	Date	-
	Range	0	2	Numeric	m
	Weight	3	3	Numeric	kg
	Wingspan	0	1	Numeric	cm
	Length	0	3	Numeric	cm
	Numbers Built	1	2	Numeric	-
	Top Speed	2	1	Numeric	km/h
Airport	Location	0	3	String	-
	Operator	0	2	String	-

	Airport Type	0	0	String	-
	Passengers	0	1	Numeric	-
	Website	0	1	AnyURI	-
Band	Origin	0	4	String	-
	Grammy Awards	2	0	Numeric	-
	Website	0	1	AnyURI	-
Car	Wheelbase	0	0	Numeric	mm
	Curb Weight	3	4	Numeric	kg
	Top Speed	2	2	Numeric	km/h
	Horsepower	0	2	Numeric	HP
	Torque	0	2	Numeric	Nm
	Length	0	3	Numeric	cm
City	Time Zone	0	2	String	-
	Altitude	1	2	Numeric	m
	Annual Rainfall	0	1	Numeric	mm
	Area	0	9	Numeric	m^2
	Population	0	9	Numeric	-
Comp. Mouse	Dimensions	0	1	String*	-
	Wireless	0	0	Boolean	-
	Optical	0	0	Boolean	-
Country	Currency Code	1	3	String	-
	Capital	0	4	String	-
	Largest City	0	3	String	-
	Area	0	9	Numeric	m^2
	HDI	0	0	Numeric	%
	Population	0	9	Numeric	-
	Calling Code	0	1	Numeric	-
	Pop. Growth Rate	0	1	Numeric	%
	Unempl. Rate	0	2	Numeric	%
	Coastline	0	2	Numeric	km
Lake	Shore Length	0	2	Numeric	m
	Depth	0	3	Numeric	cm
	Surface Elevation	0	2	Numeric	m
	Surface Area	1	2	Numeric	m^2

Mobile Phone	Dimensions	0	1	String*	-
	Display Resol.	2	0	String*	-
	Talk Time	0	1	Numeric	s
	Stand-by Time	1	0	Numeric	s
	Camera Resol.	0	2	Numeric	-
	Weight	3	4	Numeric	g
	Internal Memory	0	1	Numeric	Byte
	Display Size	0	0	Numeric	m
	Wifi	2	2	Boolean	-
	Bluetooth	0	2	Boolean	-
	Card Slot	0	1	Boolean	-
	Infrared Port	0	0	Boolean	-
	USB	0	2	Boolean	-
	EDGE	0	0	Boolean	-
Movie	Director	1	5	String	-
	Writer	1	4	String	-
	Aspect Ratio	0	1	String*	-
	Cinematography	0	0	String	-
	Genre	1	5	String	-
	Release Date	0	3	Date	-
	Budget	0	2	Numeric	$
	Runtime	1	4	Numeric	s
Newspaper	Publisher	1	2	String	-
	Editor	2	3	String	-
	Owner	0	2	String	-
	Founded	0	0	Date	-
	Circulation	0	2	Numeric	-
	Website	0	1	AnyURI	-
Politician	Birth Name	1	1	String	-
	Place of Birth	3	3	String	-
	Date of Birth	2	2	Date	-
	Nationality	0	2	String	-
Restaurant	Headquarters	0	1	String	-

	Employees	1	2	Numeric	-
	Revenue	0	1	Numeric	$
	Website	0	1	AnyURI	-
Sports Team	Manager	0	2	String	-
	Coach	1	1	String	-
	President	0	1	String	-
	Founded	0	1	Date	-
	Website	0	1	AnyURI	-
University	Motto	0	3	String	-
	President	0	1	String	-
	Mascot	0	2	String	-
	Location	0	3	String	-
	Established	0	0	Date	-
	Website	0	1	AnyURI	-
	Employees	1	2	Numeric	-

Table 7.2: Fact Extraction Evaluation Ontology

7.6.2 Methodology

To evaluate the fact extraction component, we let WebKnox retrieve the top 10 ranked pages for each query using the Bing Search API. A human interpreted the extracted results and classified them as either correct, almost correct, or wrong. The following guidelines were used when judging the extractions:

- **Correct**: The extracted value is correct in the sense that one or multiple trustworthy websites confirm the fact. "Trustworthy" describes any page that seems to be reliable or an expert on the topic. For example, imdb.com is considered a trustworthy resource for movie facts, the CIA World Factbook is considered trustworthy for country data, and Wikipedia is considered trustworthy in general. It is also important to mention that some facts do not have a single absolute ground truth; for example, the price of a mobile phone cannot be stated correctly by any source as it changes constantly and depends on the merchant as well.
- **Almost Correct**: The extracted value is correct, but less precise than the optimal answer. For example, the foundation date of a sports team can sometimes be stated exactly with date, month, and year. If only the year is found, the answer is still correct, but not precise enough. Thus, the answer is considered almost correct. The same applies to person names when a middle name is omitted, for instance. For numeric facts, we consider values almost correct if they come from sources that are not trustworthy, but close to the values (about 10 % margin) found on trustworthy pages. We also consider

absolutely correct values to be only almost correct if their trust values are not higher than those of the less correct values. We speak of "almost precision" or short a-precision when we mean the precision that includes almost correct fact values.

- **Wrong**: The value is either completely wrong or could not be confirmed by a trustworthy source.

In the following sections, we use the well-known information retrieval evaluation metrics precision, recall, and F1. We calculate precision once with only correct values and once with the correct and almost correct values combined. Recall and F1 use the almost correct values in all following evaluations. Additionally, we evaluate how often the correct (or almost correct) value is extracted in the top five extractions.

7.6.3 Evaluation by Concept

In this section, we evaluate the fact extraction performance across all entities and facts for each of the 17 concepts. Figure 7.7 shows the results of the evaluation including the macro-averaged performance.

The first thing to notice is that the fact extraction performance seems to be highly dependent on the concept (and attributes within that concept). While extractions for the concept *Mobile Phone* are by far the most precise with about 83 % precision, extracted facts for the concept *Sports Team* only have an a-precision of about 26 %.

In general, the fact ranking could be improved since there is a 10 % gap between the correct top five measure and the recall. That means that the almost correct value was within the top five extracted facts, but the trust value was not the highest. This ranking deficit seems to be especially problematic for the concepts *Sports Team* and *Actor* where the gap is over 30 % and over 15 % respectively.

The recall could also be improved by using several search engines at the same time and taking more than just the top 10 results per query into account. The focus of our evaluation is on precision, but on average only about six out of ten extracted facts are correct. We can, however, use the trust and introduce a threshold to improve the precision as shown in Section 7.6.6.

7.6.4 Evaluation by Datatype

In this section we evaluate how well we are able to extract the different datatypes. Figure 7.8 shows the evaluation metrics for the five datatypes.

We can see that the "String" datatype extraction yields the lowest F1 value due to its low precision of about 35 %. This result is not surprising since this datatype is the least restrictive and therefore we extract many false positives. We can greatly improve the precision, however, when we allow slightly less precise answers that are still considered valid by a human judge. In this case, the precision increases to almost 50 %.

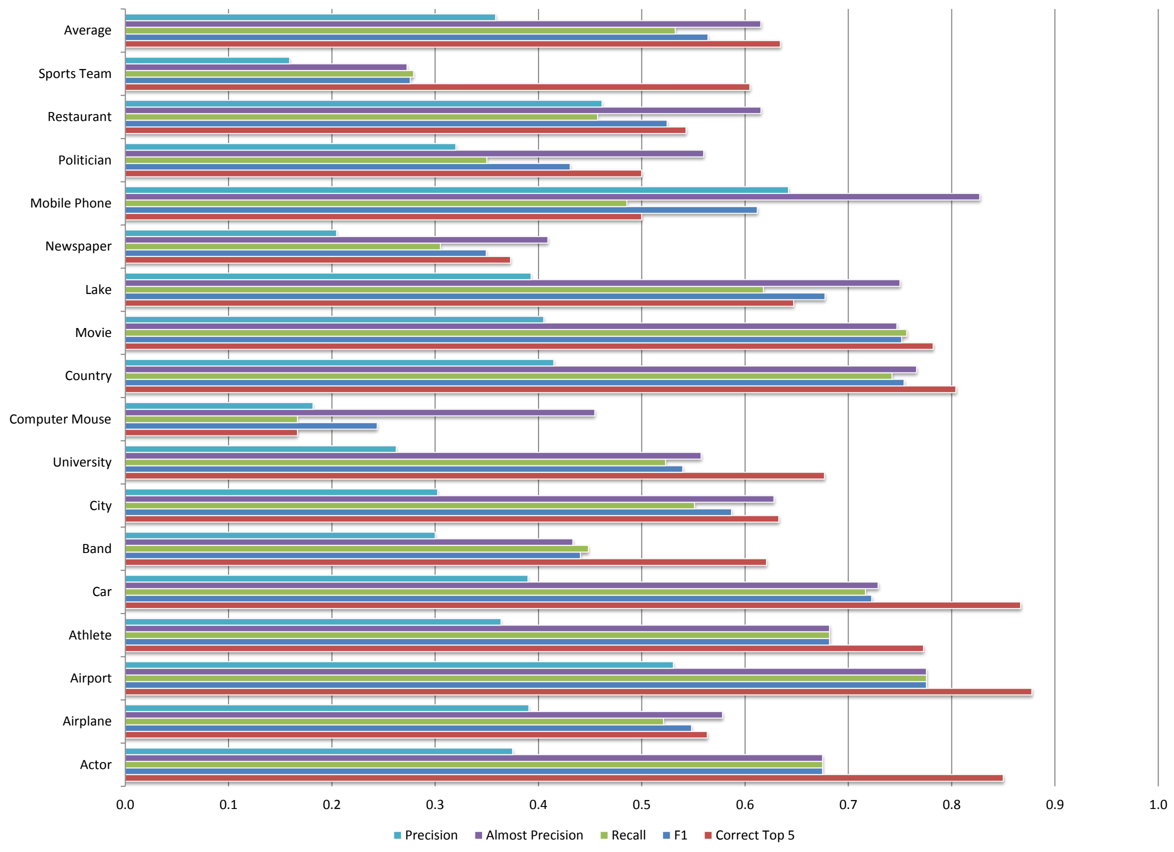


Figure 7.7: Fact Extraction Performance across All 17 Concepts

For more than 70 % of the expected "Numeric" facts, we find an almost correct answer within the top five extractions. Again, we observe a large discrepancy between precision and a-precision since many numeric values are within a range of correct values. An almost correct numeric value is often seen as perfectly correct by a human judge.

The "Date" datatype is extracted with the highest precision of about 86 % (considering almost correct dates as well). This level of precision can easily be explained since dates can be found using very specific patterns. However, we only find about 53 % of the expected dates within the top five extractions. Only a few dates were ranked incorrectly so that the correct date did not have the highest trust value. This observation becomes clear by comparing the correct top five bars (red) with the recall bars (green) in the figure.

The "Boolean" datatype can also be extracted with a rather high precision of over 80 % (including almost correct facts), though it yields the lowest recall with about 38 %. All the extracted boolean values were, however, ranked correctly so that the correct value had the highest trust (no difference between correct top five and recall). Since six of the eight boolean attributes are in the *Mobile Phone* concept, the precise extraction of those facts explains the superior extraction performance for this concept as shown in Figure 7.7.

The "AnyURI" datatype extraction leads to the highest F1 value of just over 70 %. Like dates, URIs follow very specific formats and can therefore be easily extracted.

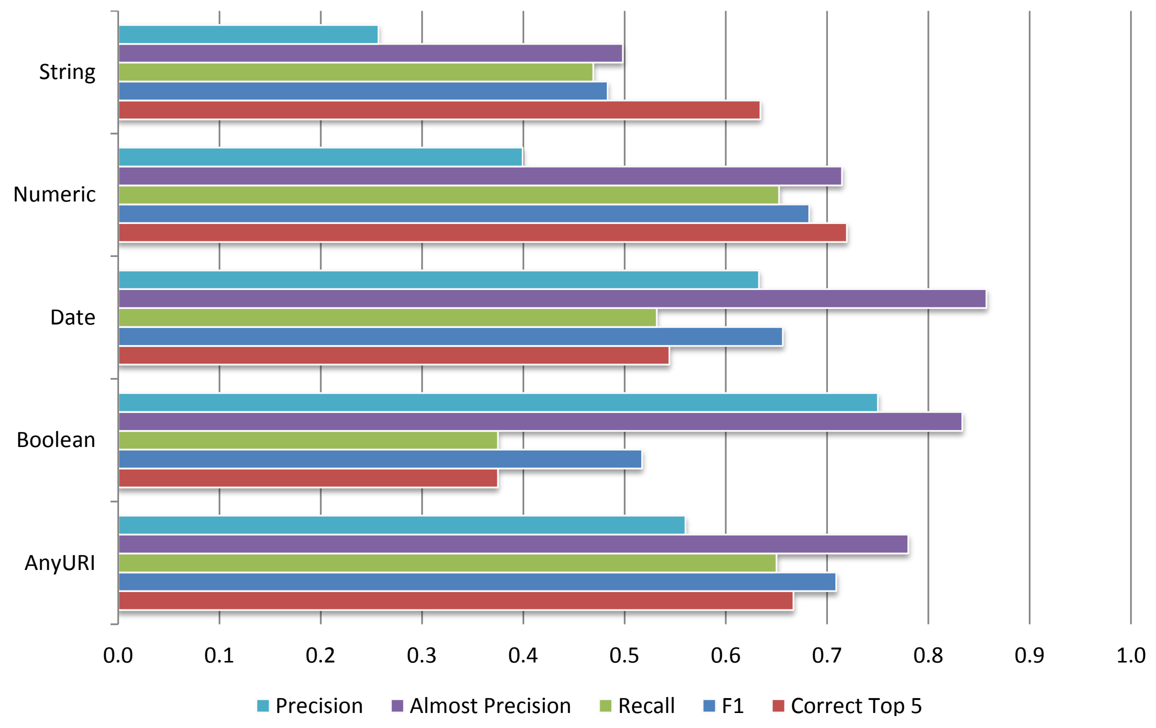


Figure 7.8: Fact Extraction Evaluation by Datatype

7.6.5 Evaluation and Comparison of Fact Extraction Techniques

In this section, we evaluate the performance of the five fact extraction techniques. Figure 7.9 shows the performance measures for each technique on the given dataset. We can see that three out of five techniques yield about the same a-precision of approximately 76 %. Plain Text and Phrase Extraction are the least precise. Extracting information without knowing the format is of course more difficult and explains the low precision of the Plain Text Extraction. The Plain Text Extraction technique, however, yields the highest recall, which is also expected since most of the information is given in plain text. Only a few of the searched facts were even found in phrases, explaining the low recall for the Phrase Extraction technique.

Clearly some techniques function better than others. Now the question is whether we need all the different techniques or whether all the facts can be found using just a subset of the five techniques. In a first analysis, we calculated the overlap of correctly found facts for all five extraction techniques using Equation 7.7 (Jaccard coefficient), where $T1$ and $T2$ are sets of correct fact extractions by two techniques. In each set, we have entity-attribute combinations for which a correct value was found. For example, $T1 = \{JimCarrey - birthname\}$ and $T2 = \{Australia - capital\}$. Interestingly, only one out of the about 1,000 expected facts was extracted by two techniques (the *manager* of the *Sports Team Real Madrid*). Other fact values were only found once by one technique. An overlap of 1.0 means that the sets are identical.

$$Overlap(T1, T2) = \frac{|T1 \cap T2|}{|T1 \cup T2|} \tag{7.7}$$

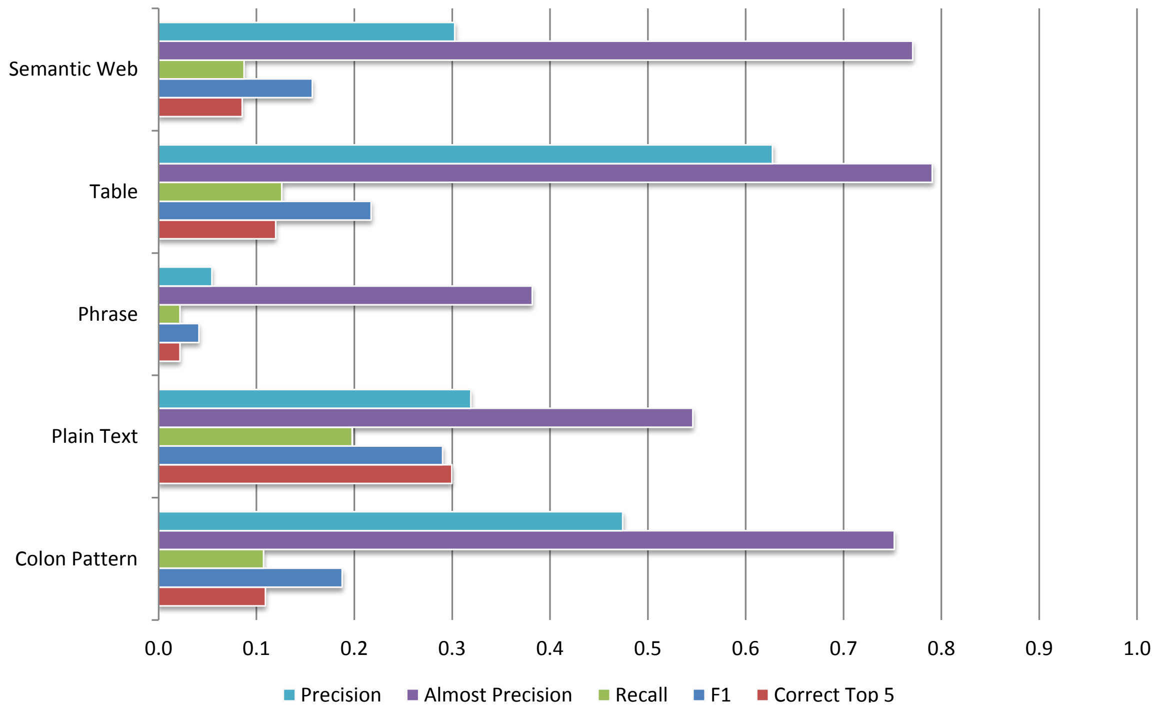


Figure 7.9: Fact Extraction Evaluation by Extraction Technique

Table 7.3 shows the overlap scores among all five techniques. The overlap is symmetric ($Overlap(T1, T2) = Overlap(T2, T1)$), hence only half the matrix is filled. We can clearly see that the techniques rarely overlap. To better visualize the overlap among all ten technique combinations, see the Venn diagrams shown in Figure 7.10. For each pair, the circles are scaled to reflect the relation of correctly extracted facts compared to the other technique. This figure also reconfirms that many correct extractions come from plain text. The largest overlap can be seen between the Phrase Extraction and the Semantic Web Extraction, that is, simple facts such as "the capital of Australia is Canberra" can often be found in phrases as well as in Semantic Web triples.

	Colon Pattern	**Phrase**	**Plain Text**	**Semantic Web**	**Table**
Colon Pattern	1	0.0067	0.0584	0.0361	0.0256
Phrase		1	0.0284	**0.1429**	0.0060
Plain Text			1	0.0492	0.0473
Semantic Web				1	0.0483
Table					1

Table 7.3: Overlap Matrix of Five Extraction Techniques

Even after calculating the overlap, we still cannot conclude whether we need all the techniques. Despite small overlaps among the different techniques, we still assume that all techniques are valuable. To test this assumption, we ran a second analysis. Figure 7.11 shows the distribution

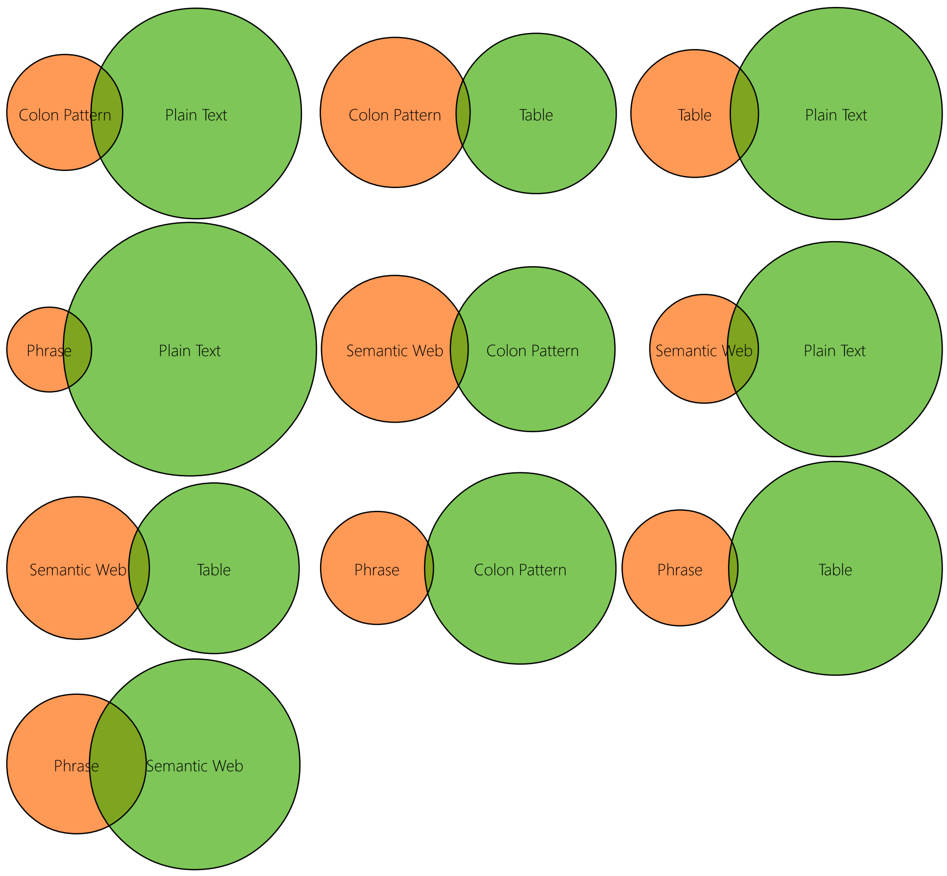


Figure 7.10: Overlap of the Fact Extraction Techniques

of correctly extracted facts by the number of techniques that found facts for entity-attribute combinations. The diagram also reveals that 46 % of the correct facts for entity-attribute combinations were **only** extracted by the Plain Text Extraction technique. Only 17 % of the facts were found using two or more extraction techniques. Even the very imprecise Phrase Extraction technique adds another 3 % of correct fact extractions that were not found by any other technique.

From this analysis, we conclude that all techniques should be used for the fact extraction process. It is important, however, to remember how facts were extracted since the techniques vary greatly in their extraction precision.

7.6.6 Trust Threshold Analysis

In Section 7.5 we have demonstrated how to calculate a trust value (*CombinedTrust*, see Equation 7.4 and Equation 7.6) for each extracted fact value. Based on this fact trust, we can discard extractions about which we are not confident. We re-evaluated the averaged a-precision, recall, and F1 values for all concepts, datatypes, and extraction techniques in dependency on the assigned trust. With rising trust values, we expect precision to increase while the recall drops. In Figure 7.12, we witness exactly this behavior. We calculated

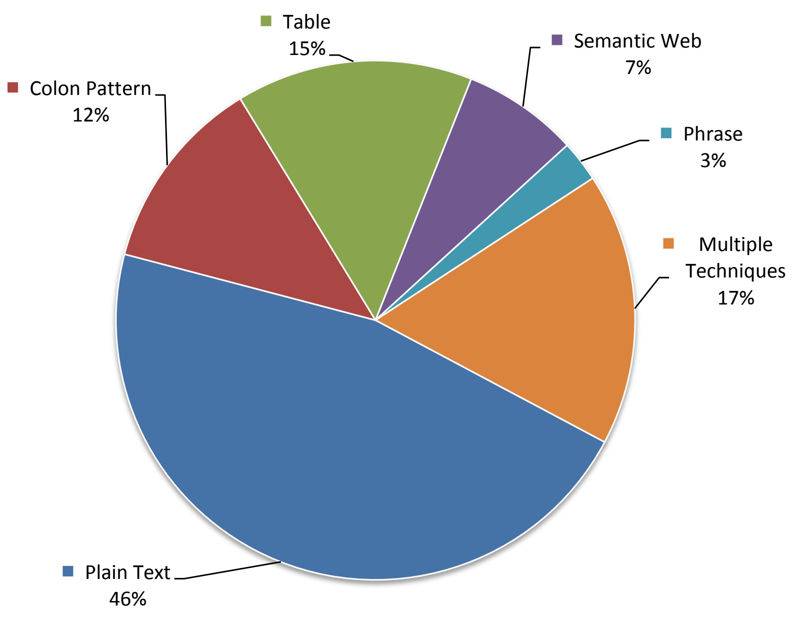


Figure 7.11: Ratio of Facts Extracted Only by One Fact Extraction Technique

the performance measures in 0.1 increments in the interval $[0, 1]$. For each trust increment, only extracted values with a CombinedTrust greater or equal to this value were used for the calculation.

We now see that we can reach a maximum precision of 74.9 % (including almost correct values) at the cost of recall, which drops to 19.3 %.

7.7 Summary

In this chapter, we have introduced the concept of fact extraction from the Web. First, we showed how the fact extraction is coupled with the entire WebKnox architecture and which steps are performed. Second, we discussed ontology engineering to create an ontology for the fact extraction process. Third, we examined related work in the field of fact extraction and explained why our approach is different and what advantages it presents. Fourth, we described the retrieval and extraction techniques, which use different sources, structures, and formats in which facts for entities can be found. Fifth, we described a set of equations for assigning a trust value to each extraction. In the last section of this chapter, we evaluated the extraction results on an ontology with over 100 attributes in 17 concepts. In conclusion, we can say that after filtering low-trusted extractions, WebKnox can correctly find every fifth fact we are looking for and about three out of four extracted facts are (almost) correct.

We have used a self-supervised fact assessment algorithm to assign trust values to the extracted facts. To improve the assessment quality, a supervised assessment algorithm, similar to the approaches in Chapter 6, could improve the overall results. Furthermore, we have seen that structured values such as dates and URIs were extracted with almost double the

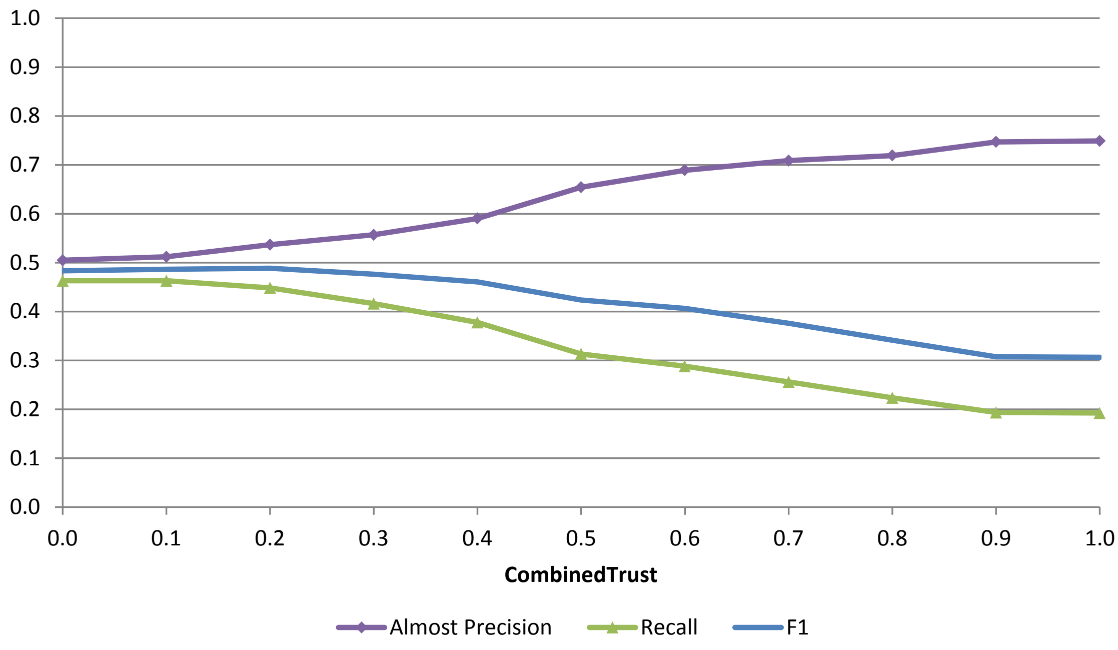


Figure 7.12: Trust Threshold Analysis of Extracted Facts

precision than facts that are just expected to be some sort of string. We therefore believe that using regular expressions to specify the expected fact value format can increase the extraction precision significantly.

Fact extraction is an important part of the complete WebKnox system. After entities have been extracted, we use a hand-crafted ontology to find facts about these entities. Having this information allows for many applications. For example, factual information could be exported as triples for the Semantic Web and linked to other datasets, or could be used for question answering, which will be discussed later.

Chapter 8

Extraction of Multimedia Objects, Events, and Statements about Entities

As we stated in the motivation of this thesis, a knowledge base of entities is beneficial in many use cases. For question answering, additional information is of importance. In our fourth research question (see Section 1.4), we ask what entity-centric information would be useful for that purpose and how we can extract it. In this chapter, we now present techniques for extracting **multimedia objects** (images, interactive objects), **events**, and **(opinionated) statements** about the extracted entities. All this information is connected to the entities in our knowledge base and can be used for the question answering use case described in the first chapter. For example, the following questions benefit from a question answering approach that has entity-centric information available:

- "What does a goat look like?" asks for an **image** of an entity.
- "Where did Barack Obama give his inauguration speech?" asks for a place which is part of an **event**.
- "What do people think of the iPhone 5?" asks for **opinionated statements** about an entity.

The next three sections are structured similarly. First, we briefly motivate the extraction of the information type (multimedia, events, and statements). Next, we review related work on the topic. We then present our own approaches and finally, we evaluate our claims in Thesis 4 and compare our algorithms to the state-of-the-art whenever possible.

8.1 Extraction of Interactive Multimedia Objects and Images

The overall goal for WebKnox is to give the user an enhanced experience of the entity for which he searches. Entities, especially products, may be better presented in an interactive form. For example, mobile phones may be presented in a simple applet[1] that enables the user to rotate the virtual phone and thereby get more visual information about the product. Another example could be a movie that the production company promotes with little games[2]. These games should be shown to the user who is looking for information about the film. Searching for those multimedia objects using multi-purpose search engines such as Google or Bing is a tedious task since the search engines do not always explicitly index these objects (Werner, 2010). We therefore need an automatic mechanism to find and extract relevant interactive multimedia objects for given entities. In this chapter, we also briefly describe an image retrieval algorithm and evaluate it across a set of 17 concepts to demonstrate how much easier it is to extract still images compared to interactive multimedia objects.

In this section we describe how we can search and extract these interactive multimedia objects which we call IMOs. "IMOs" are Web objects that combine media such as images, video, or sound. These media must have an interface for users to interact with them. The objects run in a Web browser on the client with or without additional plugins.

A simple IMO is a video that allows the user to play, pause, and control the volume. Although this fits our definition, we are more interested in strongly interactive objects, such as 3D applets where the user can rotate an object.

8.1.1 IMO Types

There are five main formats that are used to create IMOs on the Web today:

1. **Flash**: Adobe Flash applications are the most widespread type of interactive applications on the Web today. 96 % of all Web browsers have a Flash plugin (OWL, 2012) and are therefore able to show Flash applets. The files are SWF files, which can also be indexed using a special parser that Adobe developed for search engines (Adobe, 2008). Using this parser, search engine bots can simulate users and virtually click through the content of the application. This way, they are also able to read text and extract images from the Flash file. Since HTML 5 is trying to provide all the features of Flash using an open standard, Flash can now be exported to HTML 5 Canvas as well.

2. **Silverlight**: Silverlight is the Microsoft pendant for Adobe's Flash. Browsers need a special plugin in order to see the applications, but only roughly 60 % of all Web browsers have this plugin installed (OWL, 2012). The application is packaged in a XAP file, which contains an application markup file (XAML), the application itself (DLL), and additional contents such as images and videos. Search engines can also look into these XAP files, read the XML, and find the additional contents. The developer, however, decides how much information is put into these files to make it easier for search engines.

3. **Java Applets**: Java Applets are Java-based applications that run in the Java Virtual Machine on computers that have a Java runtime installed. Java is widespread – approximately 79 % of computers have a Java runtime. Still, Java applets are in the

[1]GSMArena uses little 3D applets for many of the reviewed phones: `http://www.gsmarena.com/samsung_galaxy_ace_s5830-3d-spin-3724.php`, last accessed on 25th of March 2012

[2]For example, for the movie *Ice Age* a series of games have been developed: `http://www.y8.com/games/Ice_Age_Dawn_of_the_Dinosaurs`, last accessed on 25th of March 2012

minority and are also quite difficult to index for search engines. The code is packaged in a JAR file in CLASS files. Similar to Java applets, Java FX runs on any client that can host the Java Virtual Machine. The focus of JavaFX is platform independence, so the applications can run on desktop computers, smart phones, and even some television sets.

4. **QuickTime Applets**: Apple's QuickTime is a multimedia architecture consisting of a framework, an API, and a data format. QuickTime applications can embed multimedia content and interaction features. The file formats are usually MOV, QT, and QTVR, and search bots do not have easy access to the file contents. Additionally, only about 59 % of the browsers had QuickTime installed (OWL, 2012).

5. **HTML 5**: HTML 5 is a set of new markups that aim to replace the need for proprietary applets such as Flash and Silverlight. Modern browsers already support large parts of the standard[3] and no plugin is necessary to see the applications. The most important tag in HTML 5 is the canvas, which allows a developer to draw on it. This way, features such as video playback, which is still the main application for Flash, can be done without proprietary tools. HTML 5 objects are easier to index since all the parts that make the application are on the Web server (or reachable over another one) and no proprietary formats such as Flash need to be employed. The division of the actual application into several files such as JavaScript and Cascading Style Sheets, however, also poses a problem for our purpose of finding the "object" and referring to it. An HTML 5 application is essentially an HTML page.

8.1.2 Related Work

In general, we can distinguish between two approaches in finding and extracting IMOs: context-based and content-based. Searching the context of an IMO candidate can give more information on which topic/entity is presented in the IMO. For example, when we detect an IMO on a Web page, we can read the text around the IMO to get a better understanding of what the IMO is about. WebSeer (Frankel et al., 1996) indexes images using terms found in HTML headlines, ALT-tags, hyperlinks, and other context features. The content-based approach, on the other hand, does not always work since it is sometimes difficult to search inside the IMOs content when it is delivered in a binary form. Yang et al. (2005) and Meng and Liu (2008) have shown, however, that searching the contents of Flash applets can be rewarding. They were able to find components such as images and videos, and could also detect user interaction components such as buttons and scrollbars.

Since it is not possible to perform generic content-based extraction techniques on all the main formats we have described, we use a context-based approach with only minor content-based features for Flash. We manually searched for eight entities for each of the concepts *Mobile Phone*, *Printer*, *Movie*, *Car*, and *Headphone* on Google to find out how IMOs can be found manually and which formats dominate. As a result, we discovered that Flash is the dominating format for IMOs. For all 40 entities we found at least one Flash IMO (Werner, 2010).

[3]Wikipedia keeps a compatibilty chart up-to-date: `http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML5)`, last accessed on 25th of March 2012

8.1.3 IMO Extraction

The IMO extraction component works as shown in Figure 8.1. We use the entities and their concepts from the knowledge base to formulate search queries with a vocabulary. Terms in the vocabulary are concept dependent. For example, when searching for mobile phones we use "360 view" as a search term, so the complete query that we send to the search engine might be "Samsung Galaxy S 360 view". All retrieved pages are then candidates for IMO extraction. We now look for terms that indicate an IMO format. For example, we search for `application/x-shockwave-flash` on the page and can find embedded Flash files. A complete list of IMO indicator terms were described by Werner (2010).

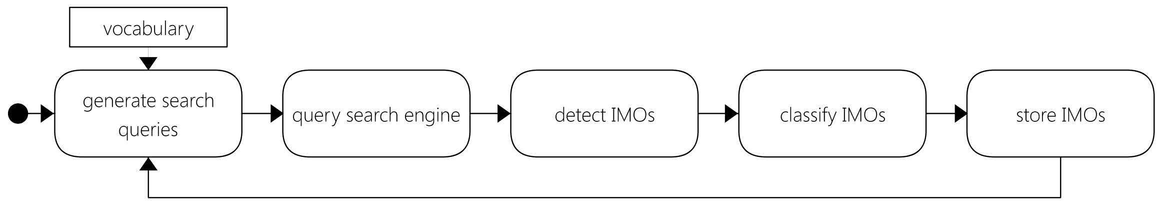


Figure 8.1: IMO Extraction Workflow

For each IMO candidate, we now analyze the context, that is, we search for the entity name in ALT-tags, link titles, the IMO file name, HTML headlines, and in surrounding text. Table 8.1 shows a list of elements that we compare to the entity name. Each comparison results in a relevance score between the element and the entity which is calculated using string similarity. For Flash applets we also decompile the file and search the text content of the file itself for occurrences of the entity name. Flash applets are often ad banners, which are not necessarily the type of IMO we want. For this reason, we discard all applets that have a typical width and height of an ad banner[4].

Feature Type	**IMO-Oriented**	**Page-Oriented**
Contextual	FileName score, FilePath score, blackList score, ALT text score, surrounding text score, XML-FileName score, XML-FileContent score	Headline score, page title score, IMO page URL score, link name score, link title score, iframe parent title score, dedicated page trust
Content	banner score, text content score	-

Table 8.1: Features for IMO Extraction

Once we have extracted a list of features, we use a trained model to calculate the IMO trust value. The last part of the IMO classification step is the interactivity classification, which is explained in the next section.

[4]These are some of the most widespread banner formats: `http://www.bannergarage.com/bannerAds.aspx`, last accessed on 25th of March 2012

8.1.4 IMO Interactivity Classification

The goal is to extract highly interactive multimedia objects, rather than videos, which only allow the user to play, pause, and control volume, for example. We therefore classify IMOs into weakly and strongly interactive. Weakly interactive is only video control, whereas strongly interactive is every additional interaction possibility.

The classification works by searching for indicator terms for weak and strong interactivity in the file name, the page title, the closest headline, and the surrounding text. Table 8.2 shows the terms that indicate weak and strong interactivity. This table has been compiled by manually analyzing Web pages with IMOs, so it can be seen as heuristics. Further investigation into which terms indicate strong or weak interactivity could be done using machine learning.

Strongly Interactive	**Weakly Interactive**
interactive, click, try, 360, view, index, main, spin, tour, virtual, gallery, play, drag, keys, game, showroom, microsite, minisite, xap, panorama	unboxing, video, preview, review, overview, movie, trailer, promotion, youtube, player, logo

Table 8.2: Indicator Terms for Strong and Weak Interactivity

For each strong interactivity term that is found in the context, we add one point to the interactivity score. For each weak interactivity term that is found in the context, we subtract one point from the interactivity score. If the final score is above zero, the IMO is considered strongly interactive, if it is below zero, it is considered weakly interactive, and if the score is zero, we have no indication of the interactivity. Formula 8.1 shows the interactivity score calculation.

$$InteractivityScore = \sum strongIndicators - \sum weakIndicators \tag{8.1}$$

Once we extract and classify the IMOs, we assign the URL of the IMO to the entities they belong to and store them as facts in the knowledge base.

8.1.5 Image Retrieval and Extraction

Not only interactive multimedia objects, but also images can interest the user. WebKnox is able to search for images and assign the URL of the image to the fact as an answer. In the knowledge ontology, one can specify how many images should be retrieved for the attribute, which is then taken into account during the retrieval process.

First, WebKnox queries the image index of a search engine with the entity name and – if specified in the ontology – the attribute name for which images should be found. For example, for the *flag* attribute for the entity *Australia* of the concept *Country*, the following query is built: "Australia flag". When searching for an image for the entity *Jim Carrey*, on the other hand, only the entity name is used in the search query.

WebKnox now retrieves a list of images returned by the search engine. It does not only rely on the first results. Instead, the goal is to find the image that best represents the sought attribute and entity. It is assumed that images that appear several times in the results are the best matching images. For example, when searching for *Jim Carrey*, some pictures appear more often because they are more widely spread on the Web and are probably more relevant.

Next, the retrieved list of images is checked for duplicates; the more duplicates a certain image instance has, the higher it is ranked. If two images have the same number of duplicates, WebKnox favors the image with the higher average position from the search engine. WebKnox then takes the top ranked images (depending on how many images are sought for the attribute and entity) and assigns the URLs of the images to the attribute.

To find duplicates, WebKnox uses an algorithm that detects differences among the images. Other algorithms such as the mean square error (van der Weken et al., 2002) or the Minkowski distance (based on mean square error) (van der Weken et al., 2002) have been tested but did not perform very well. Our duplicate checking algorithm for two images works as follows:

1. The two images are scaled to the same width. Often images with the same content are found in different sizes on the Web and must therefore be normalized. We assume that the ratio of duplicate images must be nearly the same, that is, a 250 x 100 pixel image is not likely to be the same as a 250 x 250 image.
2. If the images have almost the same ratio, both images are transformed into gray scale and a difference image is calculated by taking the absolute differences of the gray values for each pixel in both images. The more similar the images, the darker the resulting difference image. If both images are exactly the same the difference image must be black.
3. The average gray value of the difference image is calculated. If the average gray value is below a certain threshold, the images are considered identical.

8.1.6 Evaluation

In this section, we evaluate the IMO extraction component and the performance of the image extraction. For the IMO component evaluation, our evaluation measures will be precision, that is, the percentage of correctly found IMOs, and the absolute number of IMOs found. We cannot use recall as a measure since we apply all techniques on the Web and do not know the total number of relevant IMOs for our entities. For the image extraction evaluation, we use the same ontology of 17 concepts that we have used in the fact extraction evaluation (compare to Section 7.6).

Test Set

Our test set consists of two parts, a base test set and an extended test set. For the base test set, we created a vocabulary of search terms to allow the query generation process to create more focused queries for a search engine. We did use a common vocabulary for all the concepts in the extended test set to find out how domain dependent the approach is. The base test set consists of the following five concepts with seven entities in each: *Mobile Phone*, *Printer*, *Headphone*, *Movie*, and *Car*. The extended test set consists of the following four concepts with five entities in each: *Organization*, *Sight*, *Person*, and *Country*. Additionally, we added the concepts *Computer Mouse*, *Digital Camera*, and *Electronic Gadget* with five entities each to the extended test set. These additional three concepts are related to those in our base test set, so the vocabulary should work similarly for them. Overall, the total test set contains 12 concepts and 110 entities.

Manual Search vs. WebKnox

To evaluate how well the IMO extraction component works, we need to establish a baseline for comparison. We decided to compare our system to the manual approach since there are no state-of-the-art systems with similar goals available for evaluation. We therefore performed a manual search for all entities in the test set using Google. We limited the time for searching to ten minutes per entity since we assume that users would not spend more time searching for this kind of information.

WebKnox searched with six queries on average (dependent on the vocabulary) for each of the 110 entities, taking only the top five results returned from the search engine for each query. We will now compare the results of the manual search and WebKnox with regard to precision and the total number of relevant IMOs. "Match" is the number of IMOs that were in both sets, the manually found IMOs and the IMOs found by WebKnox. As shown in Figure 8.1, the IMO classification process uses features on the Web page and in the IMO to calculate a trust value, which represents its confidence that the extraction is in fact an IMO. We compare the results with and without this trust filter. To filter out irrelevant IMOs, we need a threshold for the trust value. We found this threshold by searching IMOs for 12 sample entities. We then manually classified the IMOs as relevant or irrelevant. The manual classification resulted in a set of 371 ratings, which were used to determine a threshold where the harmonic mean between precision and recall had a peak. This peak was found at a trust value of 30.2 %.

Figure 8.2 shows the evaluation across all 12 concepts. The purple bar (4th bar) shows the precision and the orange bar (6th bar) shows the absolute number of the extracted IMOs before the trust filtering. The turquoise bar (5th bar) shows the precision and light blue bar (7th bar) show the absolute number of extracted IMOs after the trust filtering. The blue bar (1st bar) shows the absolute number of relevant IMOs that were found when searching manually. The green bar (3rd bar) shows how many relevant IMOs were found by WebKnox and the red bar (2nd bar) shows how many of these overlap with the IMOs that we found manually. In general, we can observe that WebKnox finds many more relevant IMOs than we were able to find manually. Also, the relevancy in all concepts went up after the trust filtering while avoiding many false negatives.

Comparison with Google Filetype Search

In this experiment we compare the IMO extraction component with the most similar state-of-the-art system – the Google filetype search. Google indexes SWF files and we can search for them by adding `filetype:swf` to a entity name in the query. This limits the results to Flash

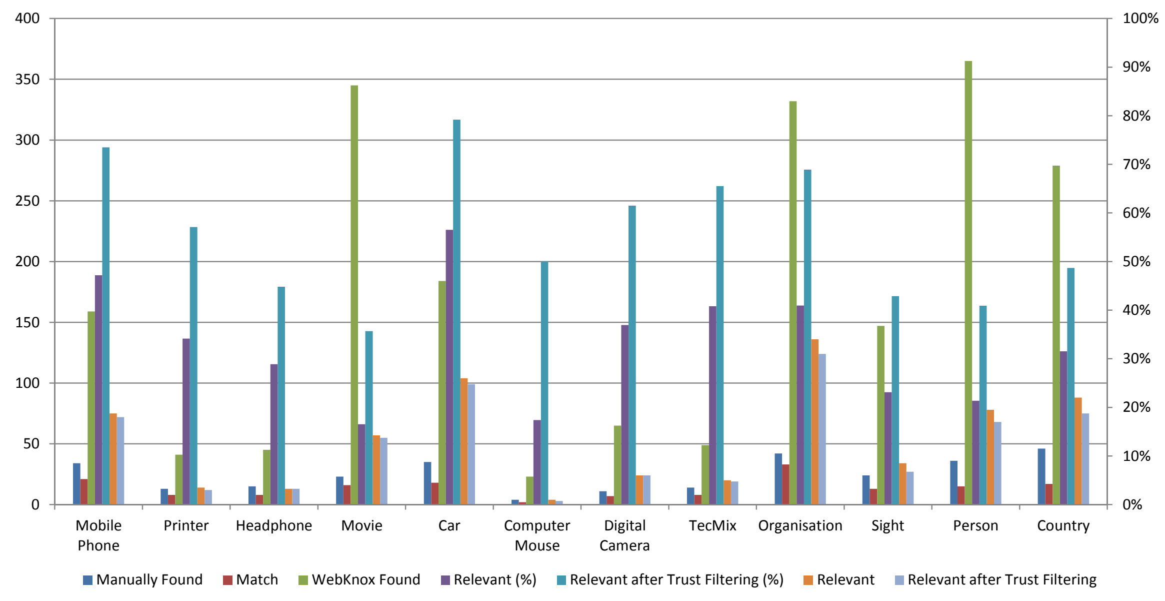


Figure 8.2: IMO Extraction Evaluation

files, but in our evaluation we found that about 90 % of multimedia files are in fact Flash files. We compare the top 10 search results from Google against the top 10 from WebKnox. In order to compare fairly, we ranked our IMO extractions by their trust value and took only the top 10 IMOs. Figure 8.3 shows the absolute number of relevant IMOs for Google (blue) and WebKnox (green). The red bars show the number of relevant IMOs found in both sets. We used our basic test set with five concepts and seven entities each. The maximum number of relevant IMOs per concept for the top 10 list is therefore 70.

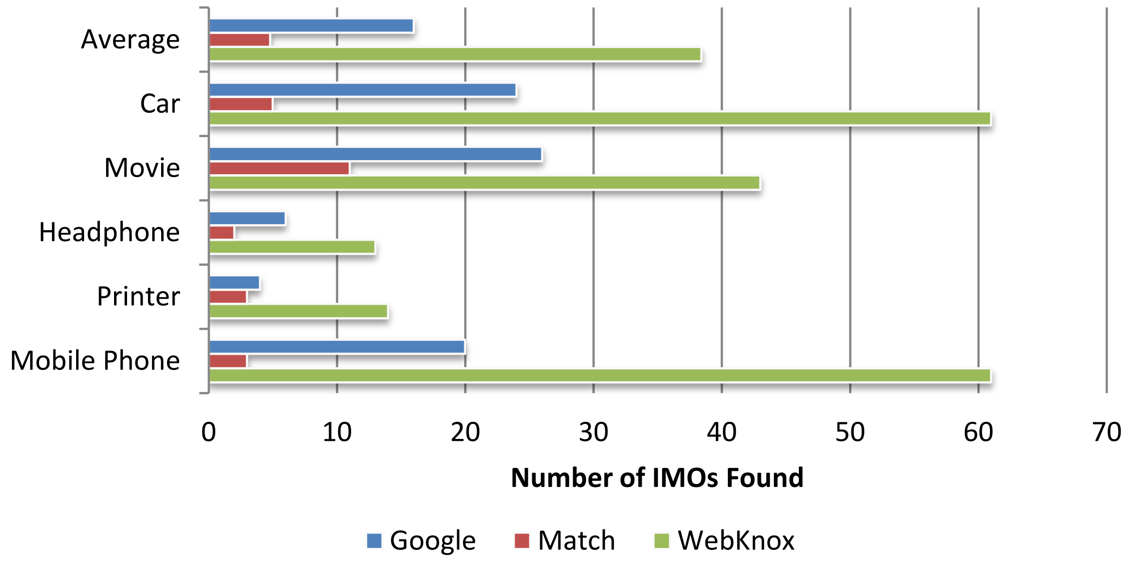


Figure 8.3: WebKnox IMO Extraction versus Google Filetype Search

On average, we were able to find 2.55 times more relevant IMOs than Google in the top 10 results. Interestingly, the match between the filetype search and our results is quite small with only 10.26 % on average. The low match percentage shows that we might need to consider adding a filetype search approach to our IMO extraction component. We have not yet tested this search approach since we want to look for different IMO types on multi-purpose search engines and only Google supported the filetype search for SWF files at the time of the experiment.

Interactivity Classification

In this section, we evaluate how well the interactivity classification of the IMOs worked. We used both the basic and the extended test sets. We extracted 647 IMOs and were able to correctly classify 277 (about 42.8 %) as weakly or strongly interactive. We were not able to classify 224 IMOs (about 34.6 %) and left them "unknown". The remaining 146 IMOs (about 22.6 %) were incorrectly classified. Figure 8.4 shows the distribution of the classifications. Ultimately, the precision of the interactivity classification is 65.48 %, the recall is 42.8 % and the F1 value is 51.76 %. These results are not satisfactory, but classifying the level of interactivity is of minor importance compared to finding the relevant IMOs. More work on the vocabulary terms for judging the interactivity level is needed and a higher precision can probably be achieved by analyzing content-based features of Flash files.

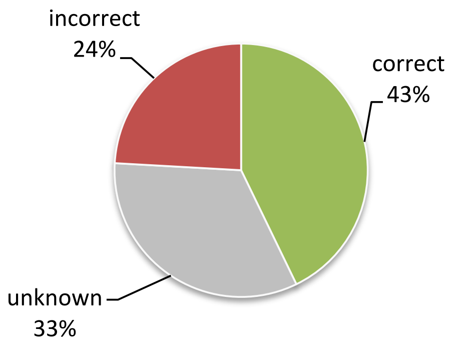


Figure 8.4: IMO Interactivity Classification Evaluation

Image Extraction

For the image extraction evaluation, we used the same test set of 17 concepts used in the entity and fact extraction evaluations. Figure 8.5 shows the precision, (almost) precision, recall, and the F1 value across all 17 concepts. We expected four images per entity for a total of 680 images. We considered an image to be correct if it was unique and depicted (part of) the entity. All images that were duplicates or only partially related (for example, a picture with several people including the person for whom we were looking) were considered almost correct.

We can see that the F1 values are between 82 % (*Computer Mouse*) and 100 % (*University*). On average, about 96.5 % of the extracted images are (almost) correct. The full recall was not reached because some image formats were not extracted correctly. The false positive extractions were almost always for rather ambiguous entities. For example, not all the extracted images for the movie *Super 8* were correct since "Super 8" is also the name of a motel chain and a film format. The film format is more of a problem since it also belongs to the *Movie* concept.

This experiment shows how much easier it is to extract still images in contrast to more complex IMOs.

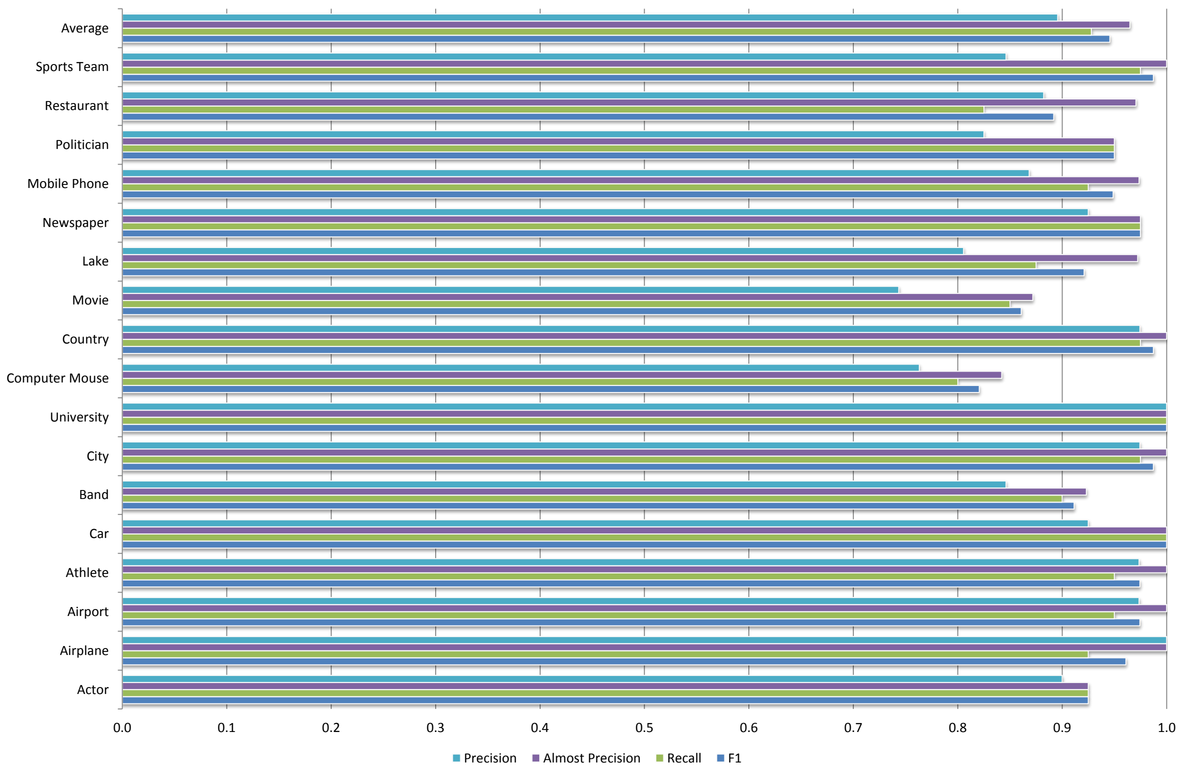


Figure 8.5: Image Extraction Evaluation across 17 Concepts

8.1.7 Summary Multimedia Extraction

In this section, we have motivated, described, and evaluated a component for extracting interactive multimedia objects and images from the Web. We were able to show that the manual search is tedious and yields a much lower number of relevant IMOs than our automated approach. We were able to find about 80 % more relevant IMOs compared to searching manually. We then compared our system to the Google filetype search, which is the best freely available system to solve the problem we face. Our extraction component finds as many as 2.55 times more relevant IMOs when comparing the top 10 results for 70 entities in our basic test set. The classification of the IMOs' interactivity works with a precision of about 65 %, leaving much room for improvement. Nevertheless, to the best of our knowledge, the WebKnox IMO extraction component is now the only system for finding and extracting interactive multimedia objects from the Web. In addition to the IMO evaluation, we evaluated entity image extraction for 170 entities across 17 concepts and reached an F1 value of about 95 %. In conclusion we can say that image extraction is much easier than IMO extraction which is mostly due to the image search support of search engines. It is, however, not (yet) possible to search for IMOs on search engines and we need to employ intelligent query and extraction algorithms. Furthermore, IMOs are by nature more diverse (formats, HTML embedding) and more complex (layout, function) than images.

8.2 Extraction of Events

In this section, we describe an event extraction component that reads news articles and extracts information related to the events in the news. In the WebKnox knowledge base, events are interesting pieces of information since they can link and put entities in context. For example, when we extract entities for the concept *Actor*, we are also interested in news about the extracted actors.

An event consists of answers to the six questions “Who”, “What”, “Where”, “When”, “Why”, and “How”, also known as 5W1H. The answers to these questions are enough to understand the event (Carmagnola, 2008). In the following sections, we will briefly review related work on event extraction, explain our event extraction approach, and evaluate our method.

8.2.1 Related Work

In the past ten years, interest in event and news summarization has spiked. Several use cases were studied and systems for disease outbreak extraction (Grishman et al., 2002) and conflict events (King and Lowe, 2003, Atkinson et al., 2008, Tanev et al., 2008) were created, for instance. In 1998, the National Institute of Standards and Technology supported the topic detection and tracking (TDT) project. The goal of TDT was to allow a comparison between systems that detect and track events in news streams over time (Yang et al., 1998, Allan et al., 1998, Yang et al., 1999). TDT ended in 2004, but event extraction research continued with the Automated Content Extraction program of the Message Understanding Conference (MUC). The goal of the MUC’s Scenario Template Task was to “extract prespecified event information and relate the event information to particular organization, person, or artifact entities involved in the event” (Marsh and Perzanowski, 1998). For each event type, there were templates that needed to be filled with related information from the text. Systems competing in this task achieved precision and recall scores between 50 % and 60 %. The templates in this task were rather rigid and could only detect prespecified event types. More complex templates capturing a wider range of event types were introduced in tasks of ACE. The goal in this task was to find mentions of events in plain text and merge them into one representation object. These representation objects hold information about entities, values, and time expressions. The ACE 2007 had only one participant in the Event Detection and Recognition (VDR) task reaching a task score of 13.4 %. This result shows how difficult it is to extract a wide range of events confidently.

Event Extraction Approaches

In general, we can distinguish three approaches for event extraction:

Pattern-based Approach Events can be captured using predefined patterns such as "in an attack, `NUMBER` people were killed in `LOCATION` on `DATE`", which would then be able to find instances of the pattern in texts such as "in an attack, 24 people were killed in Tehran on May 5th". The more specific a pattern, the higher the precision, but the lower the recall since fewer matching instance will be found. The problem with a pattern-based approach is the initial manual labor that is necessary to define good patterns.

Event-oriented Approach Events are usually reported many times in different sources. The event-oriented approaches take advantage of the different representations of the same event. Many of these approaches employ statistical features and apply clustering on the detected events to summarize news from multiple sources. Ji and Grishman (2008) use cross-sentence and cross-document evidence to detect, classify, and cluster events. This method results in higher scores in the ACE evaluation. Filatova and Hatzivassiloglou (2004), Li et al. (2006) and Liu et al. (2007) extract and organize summary sentences about events in order to have multiple sources of evidence for each event. Similarly, Naughton et al. (2008) merge news event descriptions from multiple sources into one representation. First, they identify text spans related to event mentions. Then, they link these mentions to event mentions in other documents. In a last step, they create a single coherent summary of the different event mentions. Further examples of systems using this approach are News in Essence, which uses MEAD (Radev et al., 2004), the Columbia Newsblaster (McKeown et al., 2003), and GISTexter (Finley and Harabagiu, 2002).

Semantic-based Approach More recently, Yaman et al. (2009) have applied semantic role labeling (SRL) to extract 5W1H for question answering. In SRL, different words of the sentences are labeled with their semantic roles, which could be agent, patient, source, sender, goals, good, or destination, for example. Differently-phrased sentences can have the same semantic role labels, which allows clustering of event mentions on semantic instead of syntactic representations only. In 2006, the AAAI-06 workshop on event extraction and synthesis focused on the role of semantics for event extraction. Already then, McCracken (2006) presented a promising approach combining statistical machine learning and SRL for event extraction, emphasizing the benefits of semantics.

Related Systems

In our work, we focus on extracting events by finding answers to the 5W1H. The two systems most closely related to our approach are NEXUS and CNFE.

NEXUS News Cluster Event Extraction Using Language Structures (NEXUS) (Piskorski et al., 2007) is an event extraction system that focuses on extracting violent incidents and security-related facts from online news articles. The news articles are aggregated using a media monitoring system, which already clusters news for different topics. NEXUS then uses keyword-based heuristics to detect security-related event mentions. For each mention the following actions are taken: sentence boundary disambiguation, named entity recognition, and labeling of action words and unnamed person groups.

Hand-crafted patterns, such as "killed `X`" are applied to find the events. Based on the found instances, a bootstrapping algorithm is applied to find more patterns automatically. For example, if `X` in the hand-crafted pattern "killed `X`" is instantiated with *Al Zarkawi* in a news text, another pattern "body of `X` was found" can be learned automatically.

The system's performance is very dependent on the type of extracted information. While country names can be detected with an accuracy of 95 %, cities, towns, and villages are only correct 28 % of the time. Dates were recognized with a precision of about 76 % (Piskorski et al., 2007).

CNFE The Chinese News Fact Extractor (CNFE) (Wang et al., 2010) is a news extraction system addressing the 5W1H task on single Chinese documents. They argue that SRL is computationally expensive and does not scale to large news corpora. Instead, they use a "lighter" method to extract the 5W1H. Their extraction pipeline includes topic sentence extraction, event classification, and 5W elements extraction. Interestingly, they use "Whom" instead of "Why" for one of the Ws, which is an uncommon definition of the 5W1H. This change makes the task easier since answering "Why" is often one of the hardest parts, which we will see in the evaluation later in this section.

Their extraction relies on a combination of rule-based methods and supervised machine learning using SVMs for detecting events in sentences. The 5W1H are extracted as follows:

- **Who, Whom**: Answers to these questions are identified using regular expressions with trigger rules. For example, the expression `(.*)/n(.*)/trigger(.*?)/n(.*)/n.*` matches `NP1+V+NP2+NP3`. Named entities and noun phrases are identified according to the sentence's syntactic structure before roles for each of the matches are determined using SRL.
- **What**: The answer to this question is a verb. The first verb that is classified by the SVM as "What" becomes the answer.
- **Where, When**: Outputs of the NER are taken to answer these questions. NERs work with a rather high precision for detecting locations. If no time or location entities are found, they use heuristics to find expressions that might express answers to these questions in the sentence.
- **How**: The answer to this question is a summary of the previous answers. They generate a sentence answering "Who did What to Whom" as the answer for "How".

Our approach uses different methods to extract the 5W1H. Unlike NEXUS, we do not only consider the security-relevant domain but aim to extract events from arbitrary domains. In contrast to CNFE, we focus on English news and really conform with the 5W1H instead of substituting the "Why" with "Whom".

8.2.2 Extraction of 5W1H Events

This section describes our approach to the 5W1H event extraction task. A more elaborate explanation is provided by Wunderwald (2011). The input for the Event Extractor (EvE) is a news item with its headline and content. We use a step-by-step extraction approach and apply the idea of deferred commitment (Yangarber, 2006), that is, we carry answer candidates into the next step and see whether they are still likely to be correct considering the information from the next step. Figure 8.6 shows the workflow for extracting the 5W1H.

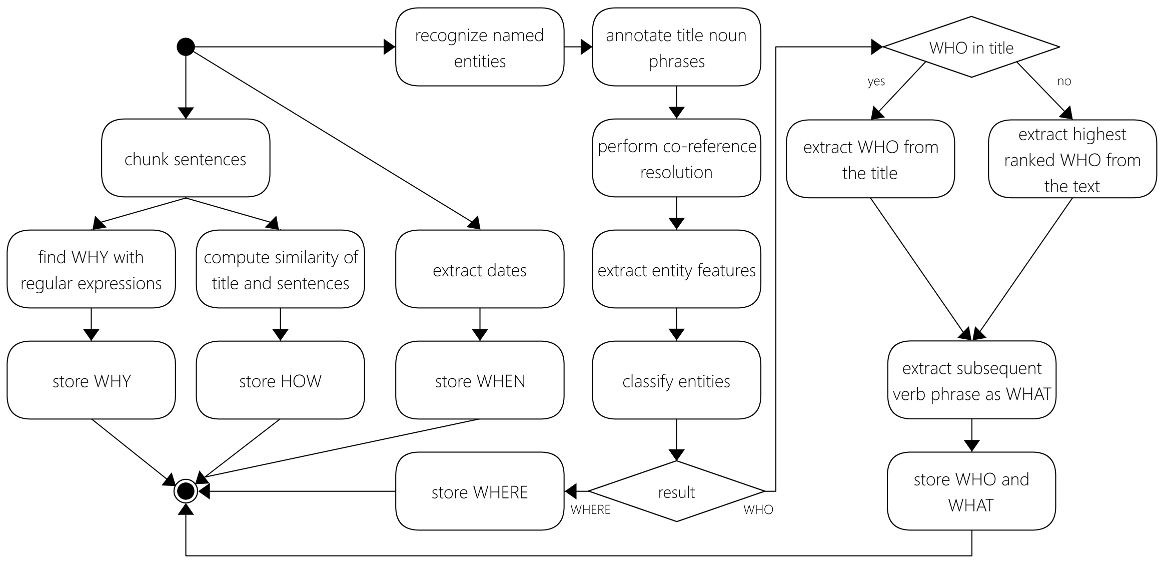

recognize named entities
annotate title noun phrases
WHO in title
yes
no
chunk sentences
perform co-reference resolution
extract WHO from the title
extract highest ranked WHO from the text
find WHY with regular expressions
compute similarity of title and sentences
extract dates
extract entity features
store WHY
store HOW
store WHEN
classify entities
extract subsequent verb phrase as WHAT
store WHERE
result
WHERE
WHO
store WHO and WHAT

Figure 8.6: Event Extraction Workflow

WHO Extraction

The WHO is the subject of an event. Often this subject is a named entity of the type *Person*, *Organization*, or *Country*. Unnamed groups can also be the subject, such as “volcano” in the sentence “The volcano erupted and damaged hundreds of houses”. In our experiments, we found that in 80 out of 100 cases, the subject is mentioned in the headline. This makes sense since the headline is often a short summary of what is written in the article. We apply NER on the article text and add noun phrases that we find in the title to the list of WHO candidates. Named entities are often mentioned several times in the text, but not always with the same syntax. We therefore apply co-reference resolution so that we can group semantically equivalent entities together (for example, *Bill Gates* and *Mr. Gates*). Each of these entity groups is a WHO candidate for which we extract the following features for the classifier:

- **Occurrences**: The number of occurrences is an indicator of how important the entity is to the given article. We distinguish between the number of occurrences in the text and in the headline.
- **Position**: The average position of the entity relative to the length of the text. The subject of the article is usually mentioned early in the article, which makes position a

good indicator for finding the subject.

- **Type**: The detected entity type from the NER.

We tested four different classifiers (Bayes networks, decision trees, bagging, and naïve Bayes) using a training set of 1,000 WHO annotations, and found that bagging works best for this task (Wunderwald, 2011). After classifying the named entities and noun phrases, we get a ranked list of WHO candidates. We store this list, but due to "deferred commitment", the ranking might still change in later extraction steps.

WHAT Extraction

The WHAT describes an action or change of state. We therefore consider verbs to be candidates for the WHAT. The WHAT is tightly coupled to the WHO since the subject carries out the action. For this reason, we search for co-occurrences between the two. If one of the WHO candidates appears in the headline, we extract the subsequent verb phrase as the WHAT. If we do not find a subject in the headline, we search for the first occurrence of the highest ranked WHO in the text and extract the verb phrase after that mention.

WHERE Extraction

The WHERE is the location where the event has taken place. Sometimes the location is irrelevant to the reported event and is not mentioned. For example, in the sentence "Barack Obama visits Germany", the location is an important part of the event, while it is not of importance in the sentence "Last Saturday, Barack Obama announced that he will run for President". We use the WHERE classifier using the same features (except the entity type) to classify location NERs. The location with the highest classification confidence becomes the WHERE of the event.

WHEN Extraction

The WHEN is the date and time that the event took place or is planned to take place. We use regular expressions to find explicit mentions of dates in the text. If we do not find any dates in the text, we assume that the publishing date of the news has the same date as the event.

WHY Extraction

The WHY is the reason that WHO did WHAT. Such an explanation is difficult to extract since semantics play a huge role in this answer type. There are, however, indicators that can be used to find likely reasons. Altenberg (1984) created a typology of causal links that he classified into four main types: "adverbial links" (for example, "hence" or "therefore"), "propositional links" (for example, "because of" or "on account of"), "subordination" (for example, "because" or "since"), and "clause-integrated lines" (for example, "that is why" or "the result was"). These words and phrases are hints for finding causal relations, but some hints are better than others. We tested the phrases from Altenberg (1984) on a set of 100 news articles and determined the accuracy of each phrase for correctly finding a causal relation. Each phrase was then used as a regular expression to extract WHY candidates. The candidate that is extracted from the expression with the highest confidence becomes the WHY of the event (Wunderwald, 2011). We found that the word "since" is the least precise indicator for WHY answers while the form "to + infinitive" after the WHAT is the most precise.

HOW Extraction

The HOW is a more detailed description of the way WHO did WHAT. Just as with WHY, it requires deep semantic analysis. Also, there is often not a single correct answer for the HOW. In our approach, we take the sentence that best summarizes the event because this sentence is most likely to be related to the WHO and the WHAT. We therefore compute the similarity of each sentence with the WHO and WHAT. The sentence with the highest similarity becomes the HOW of the event. If two sentences have the same similarity, the sentence that comes first in the text is chosen because the most relevant information has been shown to appear in the beginning of an article.

The following sentence provides a complete example of all question words: "On September 12, 2012, Barack Obama gave an eloquent speech to impress the voters in Washington D.C.". In this sentence the 5W1H are assigned as follows: WHO = Barack Obama", WHAT = "gave a speech", WHEN = "2012-09-12", WHERE = "Washington D.C.", WHY = "to impress the voters", and HOW is the complete sentence.

8.2.3 Evaluation

To evaluate the performance of the event extraction component, we let 12 users rate the extraction quality of the 5W1H for 56 news articles. Each user was given the original article, and for each question, the user had to choose whether the extraction was correct, incorrect, or partially correct. We computed a Fleiss Kappa score (Fleiss, 1981) for the user agreement of approximately 0.33 which can be interpreted as "fair agreement" (not every user rated every news which led to a modification in the score computation) (Wunderwald, 2011). The agreement among raters for WHY and HOW answers was unsurprisingly the lowest with about 0.2 and 0.25 respectively. As discussed earlier, even humans find it difficult to agree on answers to these questions.

Figure 8.7 shows the results of the user study. We can see that the WHEN answers were extracted with the highest accuracy of 75 % while the WHAT answers contained the most incorrect extractions with 36 %. As expected, the HOW and WHY have the highest proportion of partially correct answers, which, again, is due to the difficulty of agreeing on a correct answer even among human raters.

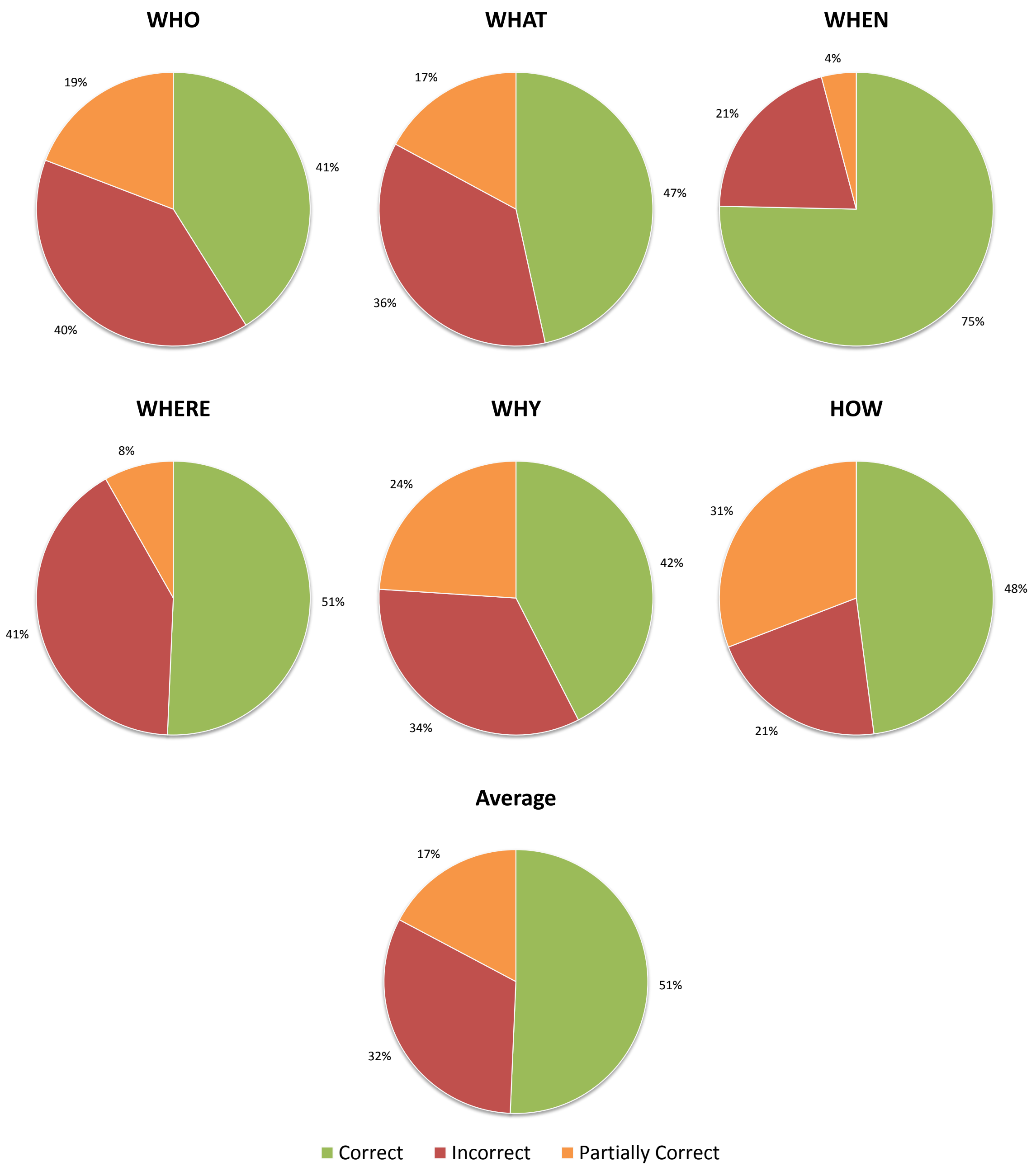


Figure 8.7: Results of the User Study for the Event Extraction Component

8.2.4 Summary Event Extraction

In this section, we described the WebKnox approach for event extraction from news articles. We have shown related work in the field and evaluated our own approach. On average, 51 % of the extracted answers were correct, and in total, about 68 % of the extractions were at least partially correct. Events are another interesting information element that can be related to entities in the knowledge base. Among many other use cases, event information can be used for question answering, which is the main purpose of the knowledge base in this thesis. For example, we could match the question “Why does Jim Carrey star in the movie The Incredible Burt Wonderstone?” to an extracted event from the sentence “Jim Carrey stars in The Incredible Burt Wonderstone because he likes director Don Scardino”, and thereby answer the question with “because he likes director Don Scardino”.

8.3 Extraction of Statements about Entities

While the fact extraction component of WebKnox searches for “common” facts, the statement extraction component searches for assertions about entities that are rarely stated in a structured form. For example, the birth place of a person is a fact that all entities from the concept *Person* must have. This information should be found with the fact extractor. A more unique fact about a person could be an event that happened to this person. For example, we might find the statement “Jim Carrey dropped out of school when he was 16”. We could not find this information with the fact extractor because it uses an ontology. The information about when a person dropped out of school is too focused, many people did not drop out of school so searching for it would not be useful. It is not a “common” fact. However, this information about an entity can be of interest so we need an approach to find and extract those entity statements.

Statements are short pieces of text (one to three sentences) that provide information about an entity. The less common this piece of information is, the more interesting and valuable the statement. Entity statements can also express opinions, which classifies them as opinionated entity statements.

8.3.1 Statement Extraction Workflow

The statement extraction component works as shown in Figure 8.8. We use the entities and their concepts from the knowledge base to formulate search queries. The queries are built following the pattern “`CONCEPT ENTITY`”, for instance, “Actor Jim Carrey”. These queries are sent to several search engines. From each Web document, we extract all sentences from the main content block of the page. That is, we seek to remove header, footer, and other navigational elements that do not belong to the content. We then take all sentences in which our sought entity appears. The last step ranks the extracted statements using a trained model for regression analysis on the list of features shown below (Friedrich, 2010).

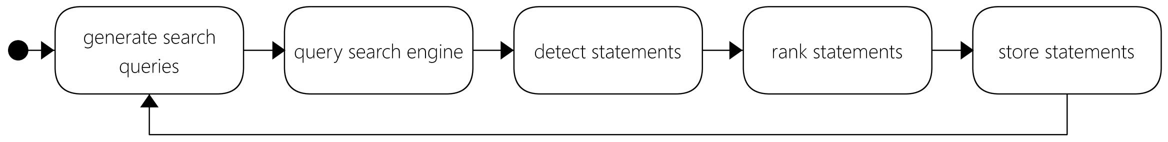


Figure 8.8: Statement Extraction Workflow

- **Search Result Position**: The position in which the Web page was found after being queried for the entity.

- **Search Engine**: The name of the search engine. Different search engines return different results, and one might be better than another.
- **Google Page Rank**: Google's page rank is a quality indicator for the page (Page et al., 1999). The rank is in the interval of [0,10] where 10 is the highest quality.
- **Top Level Domain (TLD)**: ".com" domains hold different kinds of contents than ".edu" domains, for example. We use the TLD as another feature to rank the statements.
- **Main Content Percentage**: The main content percentage measures what percentage of the Web page is in the main content block. Highly commercial sites might set up low-content pages to attract page view and money, whereas valuable Wikipedia pages have a high score in main content percentage.
- **Character Count**: The number of characters in the statement.
- **Letter Number Percentage**: The percentage of numbers in the statement.
- **Capitalized Word Count**: The number of capitalized words in the statement.
- **Syllable Per Word Count**: The average number of syllables per word in the statement.
- **Word Count**: The number of words in the statement.
- **Unique Word Count**: The number of unique words in the statement.
- **Complex Word Percentage**: The percentage of words with more than two syllables in the statement.
- **Sentence Count**: The number of sentences in the statement.
- **Words Per Sentence Count**: The average number of words per sentence in the statement.
- **Contains Proper Noun**: Whether or not the statement contains a proper noun.
- **Entity Concept**: Whether or not the concept or one of its synonyms co-occurs with the entity in the statement.
- **Entity Head**: Whether the statement starts with the entity name.
- **Related Entity Count**: The number of other entities that belong to the same concept and occur in the same statement as our target entity.

We used 200 manually scored statements to learn a linear regression model. Each statement was ranked 1 if it was relevant, 0.5 if it was somewhat relevant, and 0 if it was not relevant. The linear regression model is used for all extracted statements disregarding their entities. It could be beneficial to train a model for each concept of entities. For example, entities from the concept *Professor* might score higher from ".edu" pages, while statements about the concept *Car* might have a higher percentage of numbers due to technical data. The labor to create training data for each concept is, however, tremendous and is outside of the scope of this thesis.

To classify the sentiment of a statement, we use the text classifier that was already used for the NER from Section 5.2.4. We train a model with positive and negative statements and use this model to classify unknown statements.

8.3.2 Evaluation

Statements about entities need to be **relevant**, **interesting**, and make a reader **curious**. We measure the quality of the extracted statements in these dimensions because the goal is that human readers will find them valuable. We adapt the evaluation methodology from von Brzeski et al. (2007) which is outlined in the following paragraphs.

Methodology

Our extracted statements are presented to human judges. Each judge can rate each statement along the dimensions. Relevance and interestingness can be rated with "yes", "somewhat", "no", and "can not tell"; curiosity can be rated with "yes", "no", and "can not tell" only.

Before the evaluation, we presented each judge with guidelines. This was necessary to ensure that all judges have the same understanding of the evaluation options. We used the guidelines suggested by Friedrich (2010):

Evaluation of Relevance The relevance judgment is necessary to determine whether the statement is about the entity. The judges were told to keep the following guidelines in mind before judging the relevance:

1. Does the summary provide on-topic information for the specified object?
2. Is the summary provided in any way relevant to the object named above (in the title)?
3. Relevant are facts and opinions of any kind, as long as they are on-topic.
4. Summaries must refer to or must provide information about the object directly.
5. Summaries of websites that are talking about an object are considered somewhat relevant.
6. The object in question must be the subject of the sentence or paragraph to be relevant.

Evaluation of Interestingness The interestingness judgment is necessary to determine whether the judge found the statement interesting. To help the judges pick the right choice, they were told to keep the following guidelines in mind:

1. Do you find the information provided interesting?

2. Interesting are, for instance, facts, experiences, and opinions about the object in question, especially when unknown to the reader.
3. A summary is interesting if it is easily understood by the reader, potentially useful, novel, or validates some hypothesis that a user seeks to confirm.

Evaluation of Curiosity The curiosity judgment determines whether the reader is interested in learning more about an entity after reading the statement. It differs from the interestingness in that a reader might find the information interesting, but would not want to know more background information. The curiosity score should measure those ambitions. The following guidelines were given to the judges to help them pick the right choice:

1. Based on the summary provided, would you be interested to learn more about that content or the topic (the named object)?
2. Is the message, or part of a message, designed to arouse curiosity and interest and cause the reader to explore further, but without revealing too much?
3. If the object is underlined and clickable as a link, based on the summary around it would you actually click on it to learn more about it?

User Study Results

We extracted statements for 20 entities across four concepts. Table E.1 shows the entities used for our evaluation.

To see how our extracted statements performed compared to the state-of-the-art, we used the Google Web Search snippets that are returned for each query. Additionally, we manually searched for statements that should be relevant, interesting, and make the reader curious. We call these manually selected. For each entity and approach, we presented the top three results to the judges. We collected 1,785 judgments by 18 judges in this experiment. Figure 8.9 shows the results of the user evaluation and compares our approach to Google and to the manually selected statements (detailed numbers are shown in Table E.2 in Appendix E).

Sentiment Classification Results

We used a dataset[5] of 20,486 statements (10,593 positive, 9,893 negative, random accuracy about 52 %) that we gathered from 6,275 popular topics from Amplicate[6]. A 5-fold cross validation test with this dataset led to a classification accuracy of 89.7 %.

[5]The dataset and the results of our text classification approach can be found on Areca at `http://areca.co/16/Short-Opinionated-Sentences-about-Diverse-Topics`, last accessed on 25th of March 2012

[6]`http://amplicate.com`, last accessed on 18th of February 2012

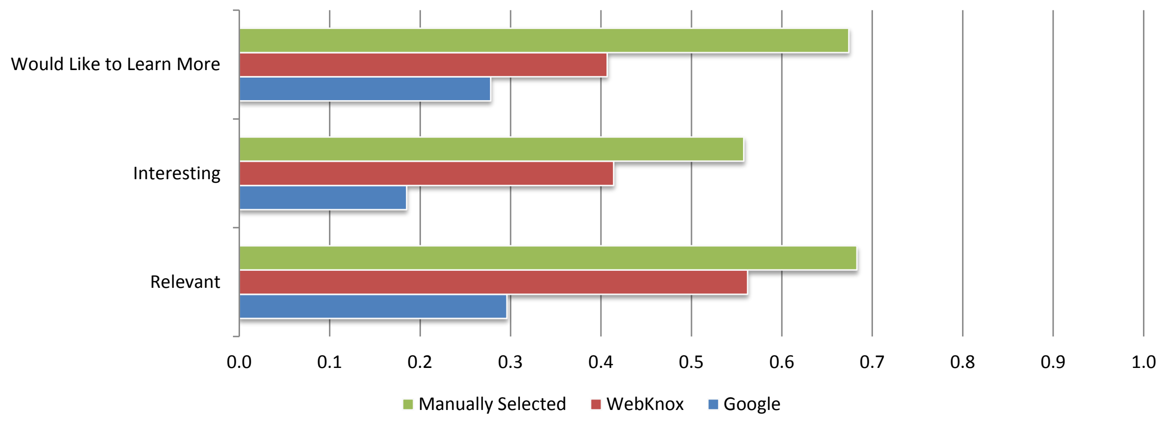


Figure 8.9: Comparison of Google, WebKnox, and Manually-selected Statements

8.3.3 Summary Statement Extraction

In this section, we have motivated, described, and evaluated a component for extracting statements about entities from the Web. The goal was to find relevant, interesting, and intriguing facts that are unique to the entity and are less likely to be found with the fact extraction component. We evaluated our system in a user study with 18 judges on a set of 20 entities from four concepts. As a comparison, we chose the Google snippets and editorially selected statements. The evaluation has shown that WebKnox surpasses the Google snippets by over 20 % on average in the measures relevancy, interestingness, and curiosity. However, there is still a huge gap between WebKnox and the editorially selected results. In future research, training a larger model or one model for each concept might narrow the gap. Another goal was to identify opinionated statements. Using a text classification algorithm, we were able to correctly separate positive and negative opinions 89.7 % of the time.

8.4 Summary

This chapter presented extraction approaches for multimedia objects, events, and (opinionated) statements about entities. We showed that using keywords and search engines such as Google, we are able to find more relevant interactive multimedia objects than we find manually. Concerning the event extraction, we showed that using machine learning for identifying parts of the 5W1H yields an accuracy of at least 51 % in our user study. We consider this result a proof of concept, but using more training data, explicit event patterns, and other hints can certainly improve event detection. Lastly, we have shown how to extract statements about entities from the Web. These statements are on average better than the snippets shown by Google under the search result. We also evaluated our hypothesis that text classification on opinionated statements yields high accuracy (see Thesis 4 in Section 1.4). In fact, we achieved almost 90 % accuracy on positive and negative sentiment statements.

We have now shown how to extract entities and fill the knowledge base with facts, interactive multimedia objects, events, and statements related to these entities. The next chapter will demonstrate how to use the knowledge base for a question answering use case.

Chapter 9

Question Answering

Our fifth research question (see Section 1.4) asks how well ontology-based question answering performs and how it compares to answer extraction approaches that operate on the Web. In this chapter, we introduce three different approaches for answering a natural language question, with and without a given knowledge base. We briefly review related work on question answering before we explain our three approaches in detail. In the final section of this chapter, we evaluate 13 different approaches for question answering on a set of real world questions.

Thesis 5 (see Section 1.4) states that ontology-based question answering systems are more precise but yield a lower recall on average. We will evaluate this thesis at the end of this chapter.

9.1 Related Work

In this section, we review the most relevant question answering systems and query intent taxonomies. We focus on question answering systems that process natural language questions instead of structured queries that can be formed using languages such as the Structured Query Language (SQL) or SPARQL Protocol and RDF Query Language (SPARQL).

9.1.1 Natural Language Question Answering

We can distinguish between three types of question answering systems: systems that **match** a user question to a known question and answer, systems that **compute** an answer by employing a knowledge base, and systems that **extract** an answer from a corpus of text, such as the Web.

Matching Systems

A simple approach toward question answering is to match a given user question to a database of already answered questions. If the system finds a good match, it presents the answers to the matching question to the user.

WikiAnswers[1], Yahoo! Answers[2], kgb Answers[3], and Mahalo Answers[4] are examples of this type of system which gets the questions and answers from an online community. FAQFinder (Burke et al., 1997) is a system that retrieves frequently asked question (FAQ) files to build a database of questions and answers. The challenge of finding and extracting FAQs using heuristics and machine learning has been studied by Jijkoun and de Rijke (2005) and sheng Lai et al. (2002) and this approach can be considered reliable with about 94 % precision and recall. More recently, Cong et al. (2008) studied the detection and extraction of questions and answers in online forums using labeled sequential patterns and machine learning. They were able to extract over two million questions with a precision of up to 88 %.

This approach can only perform well, however, if there is a large database of questions and answers to match new questions against. Most of these systems rely heavily on a community that asks and answers questions. Moreover, only questions that have already been answered can be answered again.

Computing Systems

With the rise of ontologies and Linked Data, approaches for computing an answer have become more popular. In this approach, the user question is analyzed by a query parser, which reformulates the question in order to send it to an underlying knowledge base. The answer to the question is not necessarily stored in the knowledge base already, but can be computed. For example, questions such as “How old is Jim Carrey?” or “What is the distance between Los Angeles and Chicago?” are very easy for these kinds of systems because they only require simple facts, such as the birth date of Jim Carrey or the geographic coordinates of Los Angeles and Chicago to compute the age or distance, respectively. These kinds of questions are much more difficult for systems that match questions to already answered questions since a previous user has to have asked the same question. Even if the question has been asked before, the answer might no longer be correct. In our example, the age of Jim Carrey changes, so previous answers about his age are likely outdated and therefore incorrect. Computing systems can deliver an up-to-date answer.

True Knowledge[5], Wolfram | Alpha[6], AquaLog (Lopez et al., 2007), PowerAqua (Lopez et al., 2006), Groovy Answers (Myagkikh, 2012), and the direct answers on the Google Web search page are examples of this kind of system.

This approach relies on a knowledge base of factual information that can be used to compute highly accurate answers. These knowledge bases, however, rarely contain opinions, explanations, or procedural information. For this reason, computing systems fail to answer questions such as “Why is the sky blue?” or “Should I buy an ultrabook now or wait a couple months?”.

[1]http://wiki.answers.com/, last accessed on 16th of March 2012
[2]http://answers.yahoo.com/, last accessed on 16th of March 2012
[3]http://www.kgbanswers.com/, last accessed on 16th of March 2012
[4]http://www.mahalo.com/answers/, last accessed on 16th of March 2012
[5]http://trueknowledge.com, last accessed on 16th of March 2012
[6]http://wolframalpha.com, last accessed on 16th of March 2012

Extracting Systems

The Web is a large corpus of text. Many questions a user might have are likely already answered on one of the billions of Web pages that are indexed by modern search engines. This assumption is made by great variety of systems that try to understand the question intent of the user and to find and extract an answer to the question on the Web. In general, these systems all work similarly. First, the question is analyzed. In the analysis step, questions might be translated (Zheng, 2002), part-of-speech tags might be added (Gerlach, 2012), the query intent might be analyzed, and answer patterns might be generated (Schlaefer, 2005). Second, the analyzed question is sent to one or more search engines and answers are extracted on the retrieved pages. The systems differ in how well they classify the retrieved texts as answers and rank them to match the intent of the question.

One of the earliest systems in this group is START (Katz, 1997) but many more have been created by scientists in the following years. These systems include MULDER (Kwok et al., 2001), Omnibase (Katz et al., 2002), the Boulder and the Columbia System (Pradhan et al., 2002), Tritus (Agichtein et al., 2004), NSIR (Radev et al., 2005), (Open)Ephyra (Schlaefer, 2005), Aranea (Lin et al., 2006), and AQUA (Gerlach, 2012).

The strength of this approach is that the data source – the Web – is likely to contain the answers to many questions. The weakness is that the retrieval and extraction process is resource and time consuming. Gerlach (2012) reported an average answer time to opinion-targeted questions of about 14 seconds – too long for a user to wait.

Table 9.1 classifies the mentioned systems according to their approaches. Systems with hybrid components are classified according to their main approach. For example, if True Knowledge cannot compute the answer, it tries to match the user question to kgb Answers. It is still primarily a computing system.

Matching	**Computing**	**Extracting**
WikiAnswers	True Knowledge	START
Yahoo! Answers	Wolfram | Alpha	OpenEphyra
FAQFinder	PowerAqua	AQUA
kgbAnswers	Groovy Answers	AnswerBus
Mahalo Answers	Google	MULDER
	AquaLog	OmniBase
		NSIR
		Tritus
		Aranea
		Boulder/Columbia

Table 9.1: Classification of Question Answering Systems According to their Approaches

9.1.2 Query Intent Classification

Knowing the intent of the user's query can help a question answering system find the right answer. For example, OpenEphyra tries to detect the intent of the question using a simple decision tree. If the system finds out that the user is looking for a date, it can focus its extraction process toward finding dates.

Most researchers agree on the query intent taxonomy that was introduced by Broder (2002). He classifies intents into "navigational" (the user tries to reach a certain website, such as Wikipedia), "informational" (the user searches for a piece of information, such as movie *release date*), and "transactional" (the user wants to perform an action such as booking a vacation). Later, Rose and Levinson (2004) refined this taxonomy by adding subcategories to informational and transactional intents. According to Rose and Levinson (2004), informational queries can be "directed" (the user wants to know a certain fact about a topic), "undirected" (the user wants to learn something about a topic), "advice" (the user wants to get advice, instructions, or suggestions), "locate" (the user wants to find out where something can be found in the real world), and "list" (the user wants to obtain a list about something). Jansen et al. (2008) compared user intent taxonomies proposed by different researchers and none of the compared works created more than six subcategories for the informational intent. For question answering, the available categorization proposed in research is not detailed enough. We argue that user intents can be categorized in a much more fine-grained taxonomy which supports question answering.

9.2 Question Intents

Every question has an intent. We created a question intent taxonomy that is much more fine-grained than the most detailed taxonomy found by Jansen et al. (2008). The following list describes our taxonomy, which was developed in cooperation with Gerlach (2012) by analyzing hundreds of questions on WikiAnswers and Yahoo! Answers.

- **Number**: The user wants to know a particular numeric fact. We classify all questions about "How often...", "How long...", "How much/many...", et cetera into this category. For example, the question "How many stomaches does a cow have?" belongs to this intent category.
- **Subject**: The user wants to know about the "who" of the question. For example, "Who was the 4th U.S. president?" asks for a person name. The subject can also be another entity, such as an organization or a country, as in the question, "Who is the successor of the League of Nations?".
- **Definition**: The user wants to know the definition of a term or phrase, for example, "What does xenophobia mean?".
- **Translation**: The user wants to get a term or phrase translated, for example, "What does 'gato' mean in English?".

- **Thing**: The user asks for an entity or concept, for example, "What kind of gas does the Honda Civic need?".
- **Location**: The user wants to know where something is. We distinguish between absolute and relative locations. Absolute locations include objects of the real world. The question "Where is the city Chicago?" belongs to this category. Absolute locations in the virtual world could also be intended, as in the question, "Where can I download pictures of cats?", which asks for one or multiple URLs. Relative locations, on the other hand, often require more explanation in the answer. For example, "Where can I find the volume switch on my iPhone?" or "Where is the battery in my Honda Civic?" are questions asking about a location relative to an object.
- **Date**: The user wants to know when something happens or happened, such as, "When did Hawaii become a state?". We also include date ranges in this category; the question: "From when to when was the Peloponnesian War?" therefore belongs to this category as well.
- **Procedure**: The user wants to know how something is done, for example, "How can I download YouTube videos?".
- **Difference**: The user wants to get an explanation about the difference between certain topics, for example, "What is the difference between a netbook and an ultrabook?".
- **Explanation**: The user wants to get an explanation about something, for example, "Why is the sky blue?".
- **Opinion**: The user wants to know what other people think about a certain topic, for example, "Who are you voting for in the next presidential election?".
- **Prediction**: The user wants to get a prediction about a topic, for example, "Approximately how much will the new ultrabook from Lenovo cost?".
- **TrueFalse**: Questions that only ask for a "yes" or "no" fall into this category. For example, "Is Elvis alive?" is one question with this intent.
- **LooksLike**: The user wants to know what something looks like, for example, "What does a baby goat look like?".
- **SoundsLike**: The user wants to know what something sounds like, for example, "What do singing whales sound like?".
- **Choice**: The user has two answers to a question, but does not know which is correct or better, for example, "Is Elvis dead or alive?".
- **List X**: The user wants to retrieve a list of entities or concepts, for example, "Which Asian countries are democracies?".

A question can be tagged with multiple intent categories. For example, "Where are good places to go surfing?" is asking for both an **opinion** and a **list** of **locations**. We will later see in Figure 9.8, in the evaluation section, how these question intents are distributed in different datasets.

We use a simple rule-based question intent classifier for our system. We make use of the observation that the question word often already determines the intent of the question. For example, the question word "when" is usually used when asking for a date.

9.3 Techniques for Question Answering

We now describe the question answering approach of the WebKnox system. Our approach consists of three components, one for each of the reviewed approaches **matching**, **computing**, and **extracting**. Figure 9.1 depicts the question answering process. We assume that when we have a human-written answer to a given question, it is relevant, and we use this answer. If we do not find the exact question, we try to compute an answer over a knowledge base. If we are not able to compute an answer, we use the last approach and try to extract an answer to the question from the Web. Because the extraction approach is time and resource consuming, we start it only if the other two approaches fail to answer the question.

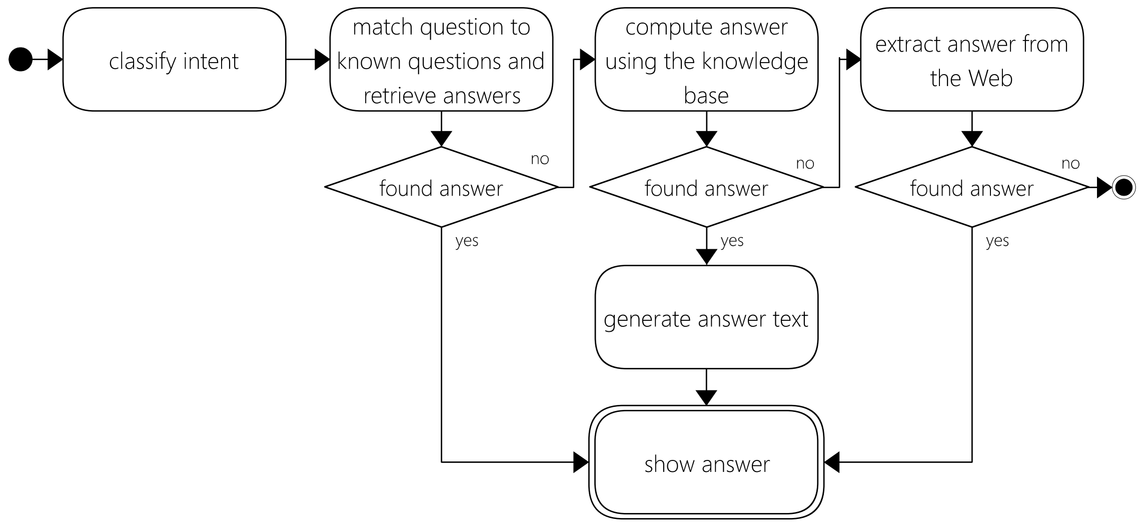

classify intent
match question to known questions and retrieve answers
compute answer using the knowledge base
extract answer from the Web
found answer
no
yes
found answer
no
yes
found answer
no
yes
generate answer text
show answer

Figure 9.1: Workflow for Answering a Question in WebKnox

The next three sections explain each of the three approaches in detail.

9.3.1 Extracting Questions and Answers from QA-rich Websites

In this section, we describe the basic design of question answer (QA) websites and how we can use it to configure the QA extractor. Furthermore, we explain our focused crawling algorithm, which increases the extraction efficiency. This extraction component makes the **matching** approach in question answering possible by providing a large database of questions and answers. We will use this database in our evaluation of the different approaches.

QA Website Design

QA-rich sites all have a similar layout, which we can use to define areas of interest. There are three possible states for a question page on a QA website:

1. There is only one question and nobody has answered it yet.
2. There is one question and one or more answers.
3. There is one question, one "best answer", and possibly, but not necessarily, other answers.

We are only interested in pages that have a question with at least one answer. To extract the question and its answer(s), we assume that every question and answer is structured using (X)HTML elements, that is, we assume that we can create an XPath (Clark and DeRose, 1999) that targets the question or answer without too much noise around it. Instead of detecting questions and their answers by lexical or semantic analysis, we take user-given XPaths to the question and answer part(s) of the page. Since the path is generated by a user and since parsing and classifying would be time and resource consuming, this is a much more effective and efficient way than using heuristics and machine learned classifiers. The downside, however, is that we need the user to find those XPath in advance. Since the number of QA-rich websites is quite small, we think the benefit of having accurate extractions is worth the additional effort.

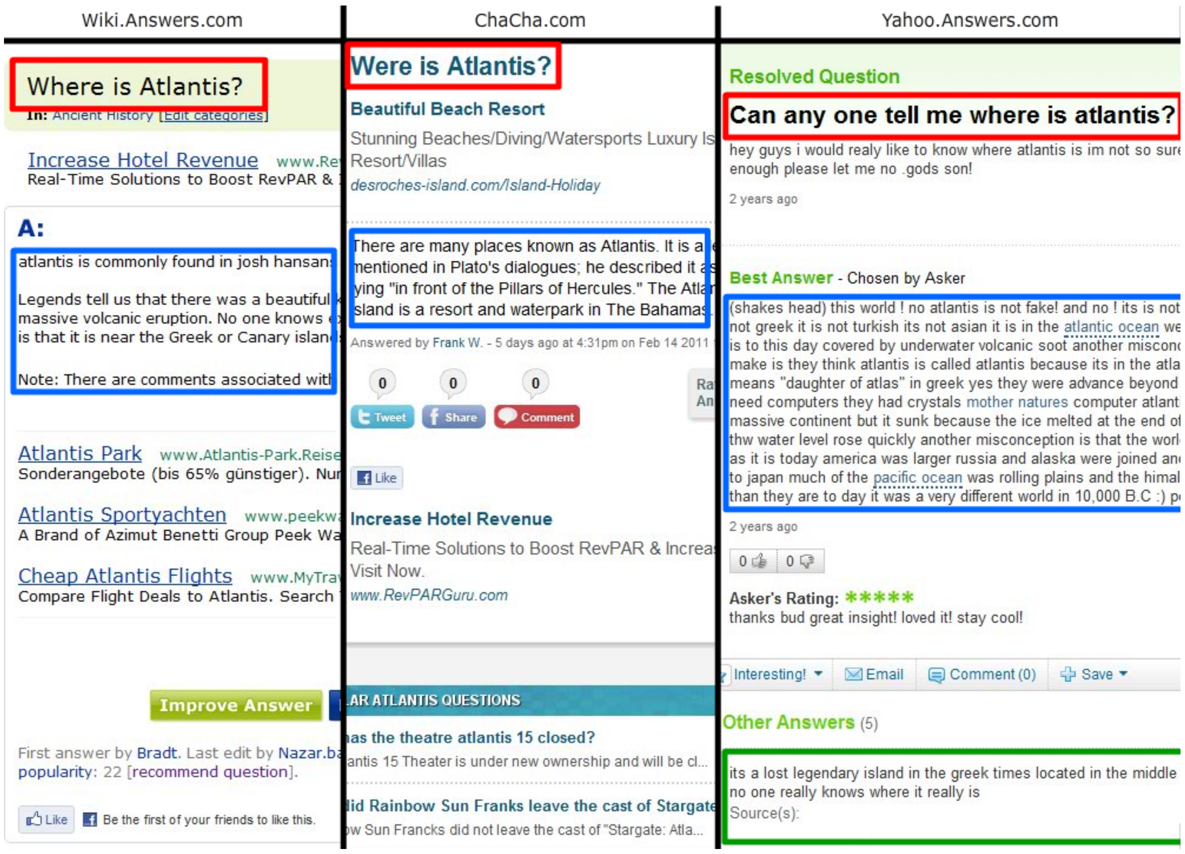
Figure 9.2 shows the layouts of question pages on the QA-rich websites Wiki Answers, ChaCha, and Yahoo! Answers. The first two are in state two, that is, the question has been answered at least once. The last one is in state three since we have one "Best Answer" and additional answers. The red frames point to the questions, the blue frames point to the best answer, and the green frame points to additional answers.

QA Configuration

We want to extract the question and answer tuples using prior knowledge about the layout of the website. A human expert must make this knowledge explicit by creating a configuration file. Figure 9.3 shows an excerpt of the file that we use to configure the extraction component. We have chosen the JavaScript object notation (JSON)[7] as a serialization format because it can be read and edited easily by human users.

We need seven parameters per page, which can be obtained in less than two minutes by an experienced user. The `name` parameter is optional and specifies the name of the website, `entryURL` specifies the start page for the crawler, `questionXPath` is the XPath that points the question section of the page (red frame in Figure 9.2), `bestAnswerXPath` points to the section with the chosen answer (blue frame in Figure 9.2), and `allAnswersXPath` points to all (additional) answers for a given question (green frame in Figure 9.2). Not every website makes a distinction between the "best answer" and additional answers, but if they do, often

[7]`http://www.json.org/`, last accessed on 18th of March 2012


Wiki.Answers.com

Where is Atlantis?
In: Ancient History [Edit categories]

Increase Hotel Revenue www.Re
Real-Time Solutions to Boost RevPAR &

A:
atlantis is commonly found in josh hansan

Legends tell us that there was a beautiful
massive volcanic eruption. No one knows
is that it is near the Greek or Canary islan

Note: There are comments associated wit

Atlantis Park www.Atlantis-Park.Reise
Sonderangebote (bis 65% günstiger). Nu

Atlantis Sportyachten www.peekw
A Brand of Azimut Benetti Group Peek Wa

Cheap Atlantis Flights www.MyTra
Compare Flight Deals to Atlantis. Search

First answer by Bradt. Last edit by Nazar.b
popularity: 22 [recommend question].

Be the first of your friends to like this.

ChaCha.com

Were is Atlantis?

Beautiful Beach Resort
Stunning Beaches/Diving/Watersports Luxury Is
Resort/Villas
desroches-island.com/Island-Holiday

There are many places known as Atlantis. It is a
mentioned in Plato's dialogues; he described it a
ying "in front of the Pillars of Hercules." The Atla
sland is a resort and waterpark in The Bahamas

Answered by Frank W. - 5 days ago at 4:31pm on Feb 14 2011

Increase Hotel Revenue
Real-Time Solutions to Boost RevPAR & Increa
Visit Now.
www.RevPARGuru.com

AR ATLANTIS QUESTIONS
as the theatre atlantis 15 closed?
antis 15 Theater is under new ownership and will be cl...
id Rainbow Sun Franks leave the cast of Stargate
w Sun Francks did not leave the cast of "Stargate: Atla...

Yahoo.Answers.com

Resolved Question
Can any one tell me where is atlantis?
hey guys i would realy like to know where atlantis is im not so sure
enough please let me no .gods son!

2 years ago

Best Answer - Chosen by Asker
(shakes head) this world ! no atlantis is not fake! and no ! its is not
not greek it is not turkish its not asian it is in the atlantic ocean we
is to this day covered by underwater volcanic soot another miscon
make is they think atlantis is called atlantis because its in the atla
means "daughter of atlas" in greek yes they were advance beyond
need computers they had crystals mother natures computer atlant
massive continent but it sunk because the ice melted at the end o
thw water level rose quickly another misconception is that the worl
as it is today america was larger russia and alaska were joined an
to japan much of the pacific ocean was rolling plains and the himal
than they are to day it was a very different world in 10,000 B.C :) p

2 years ago

Asker's Rating: *****
thanks bud great insight! loved it! stay cool!

Other Answers (5)
its a lost legendary island in the greek times located in the middle
no one really knows where it really is
Source(s):

Figure 9.2: Similar Layouts of QA-rich Websites

different XPaths must be used to extract the answers. The `answerPrefix` and `answerSuffix` parameters are used to extract the answer string without the nearby noise by specifying what must appear before and after the answer string. The extraction is then performed by retrieving the contents of the XPath for every candidate page. To clean up the answer string, the `answerPrefix` and `answerSuffix` are deleted from the string. Which pages are candidates for QA extraction is explained in the following section.

Focused Crawling

The simplest way to find all question pages in a certain domain is to crawl the complete domain from one start page. On every page, all hyperlinks are added to the URL stack and we try to find the question and answers by using the specified XPaths. This approach is a breadth-first search without any focus, and is therefore inefficient. We use focused crawling to retrieve QA pages much faster. For that purpose, we assume that the URLs that point to QA pages all have a similar prefix. For example, the two QA pages `http://wiki.answers.com/Q/What_is_the_speed_of_light` and `http://wiki.answers.com/Q/Why_is_the_sky_blue` have the common prefix `http://wiki.answers.com/Q/`. This assumption is likely to be true for many QA-rich websites, since the pages are generated with database contents and templates. To focus the crawler, we classify every page in a given domain into one of three

```
{
   "sites":
     {
        "name":              "Yahoo! Answers",
        "entryURL":          "http://answers.yahoo.com",
        "questionXPath":     "//div[1]/div[1]/div[2]/h1",
        "bestAnswerXPath":   "//div[1]/div[1]/div[2]/div[2]/div[1]",
        "allAnswersXPath":   "//div[3]/div[2]/ul/li/div/div[2]/div[1]",
        "answerPrefix":      "ratings ) ",
        "answerSuffix":      " Comments Sign in"
      }
}
```

Figure 9.3: Configuration File for Extraction from QA-rich Websites

categories:

1. **Green**, that is, a question and an answer are found on the page.
2. **Yellow**, there are no QAs on the page, but it directly links to “green” pages with QAs.
3. **Red**, there are no QAs on the page, and it is unknown whether the page links to “green” pages.

For the green category, we learn one common prefix and the number of `/` separations (address depth) in the URL. For the yellow and red categories we learn a set of common prefixes for each and also store the address depth for common prefixes from the red category. The classification is done using prefix filtering which works as shown in Figure 9.4. A Web page is taken from the URL stack (the first page is the `entryURL` as specified in the configuration file). All links to other pages within that domain are added to the URL stack. Then we search for a question and an answer and extract them using the specified XPaths. If no question is found, the page is added to the red category. If a question is found, the common prefix for green pages is updated with the current URL. Its parent URL, the URL that linked to the current page, is classified as yellow. In the next iteration, a green category URL is retrieved from the stack, that is, the prefix of the URL from the stack must match the learned common prefix for the green category. If no green page is found, we attempt to find a page from the yellow category since we know that these pages point to more green pages. If there are neither green nor yellow URLs in the stack, a URL that is not from the red category is taken from the stack. We do this to get an unclassified URL since we know that URLs from the red category definitely do not contain questions and we should explore another URL path. After only a few analyzed URLs, the prefixes for all three categories are learned, and pages containing QAs (green category) are preferred over yellow and red pages. This leads to more extracted questions after the same amount of time while the recall stays the same compared to non-focused crawling.

The approach presented here is novel with regard to the extraction source. There has been no previous work on extracting questions and answers from QA-rich websites.

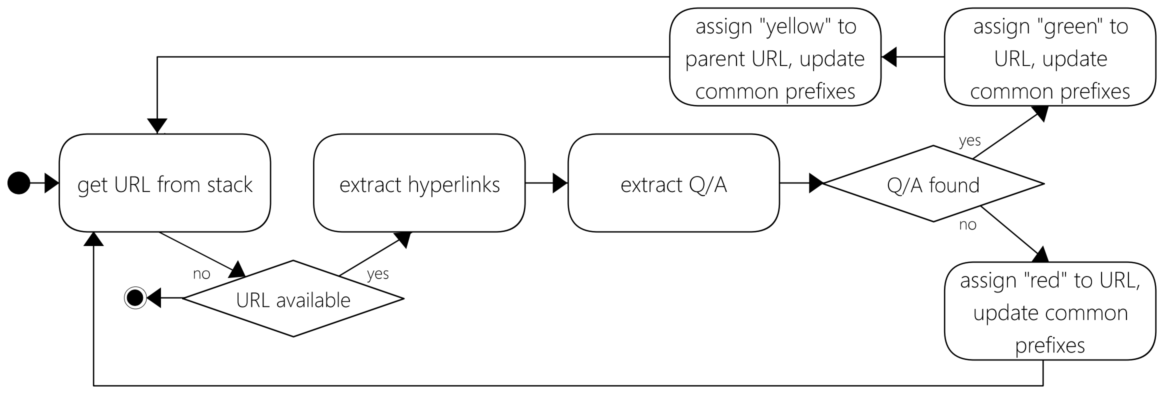

assign "yellow" to parent URL, update common prefixes
assign "green" to URL, update common prefixes
get URL from stack
extract hyperlinks
extract Q/A
Q/A found
yes
no
URL available
no
yes
assign "red" to URL, update common prefixes

Figure 9.4: Learning Process for URL Classification in Three Categories

9.3.2 Computing Answers on a Knowledge Base

This section explains an approach to answer simple factual questions using a knowledge base of RDF triples. Figure 9.5 outlines the process of the answer **computation** component.

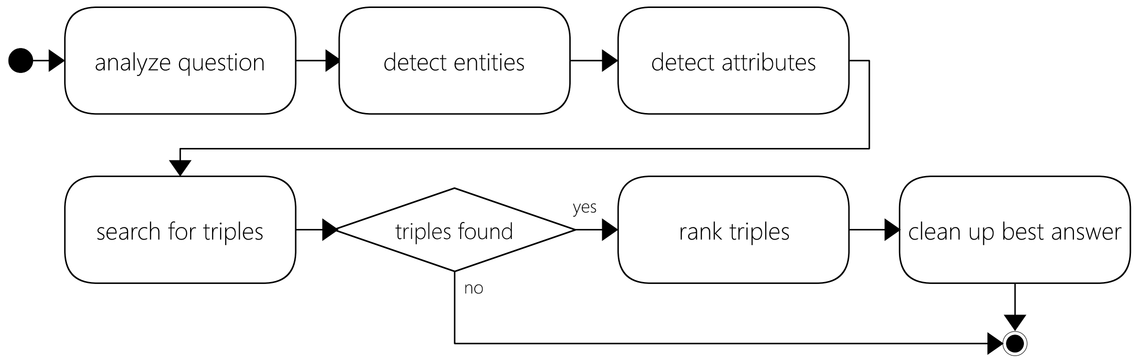


Figure 9.5: Process of Computing an Answer Using a Knowledge Base of Factual Information

First, the question is analyzed and prepared for the next steps. Next, we try to find entities in the question. To do this, we build word n-grams from the questions with $1 \leq n \leq 3$. Each of the n-grams is a candidate for an entity. We then search for entities in the knowledge base that match one of the n-grams. For example, for the question “Where was Jim Carrey born?”, we find the 2-gram “Jim Carrey”, which we then find in the knowledge base. The same step repeats with the detection of attributes. Additionally, we try to find attribute synonyms and alternative spellings in the database but lower the trust in these additional search terms. For instance, in our example question, we would find “born” to be a synonym for the attribute *birth date*. Once we have found an entity and an attribute, we search for triples that contain both. The value we are searching for can be either the subject or the object of the triple. We therefore search with the following two queries, where the question mark symbolizes the answer to our question:

1. `<Entity> <Attribute> <?>`. For example, `<Jim Carrey> <birthdate> <?>` would

instantiate `<?>` with `1962-01-17`.

2. `<?> <Attribute> <Entity>`. For example, `<?> <founded> <Microsoft>` would instantiate `<?>` with `Bill Gates`.

If we find more than one matching triple to our query, we rank the resulting triples by trust values. The best-ranked answer is then cleaned up and the process ends. If no triple is found the process ends immediately.

9.3.3 Extracting Answers from the Web

In this section, we describe an **extraction** component that searches for answers to given questions using search engines such as Google and Bing. In contrast to the focused crawling approach from Section 9.3.1, this approach is not limited to QA-rich websites. It can use any Web page containing possible answers to the given question. In a first experiment, we selected 87 random questions and asked Google for answers. For 97 % of the questions, we were able to find a correct answer on one of the pages of the top ten results; in 81 % of the cases, it was already the first result given by Google (Gerlach, 2012). This experiment shows that using search engines and their ranking for Web pages could be a viable approach for answer extraction.

Figure 9.6 illustrates each step in the answering process. First, the question is analyzed and answer patterns are created. We send these answer patterns to search engines before we try to match them on the textual content of the retrieved Web pages. If an answer is found, we clean up the answer and the process ends. If we are not able to detect an answer using a pattern, we send the original question to search engines and generate answer candidates from the retrieved Web pages. If we find candidates, we rank them and clean up the best-ranked answer before we end the process. We explain each of these steps in more detail in the following sections.

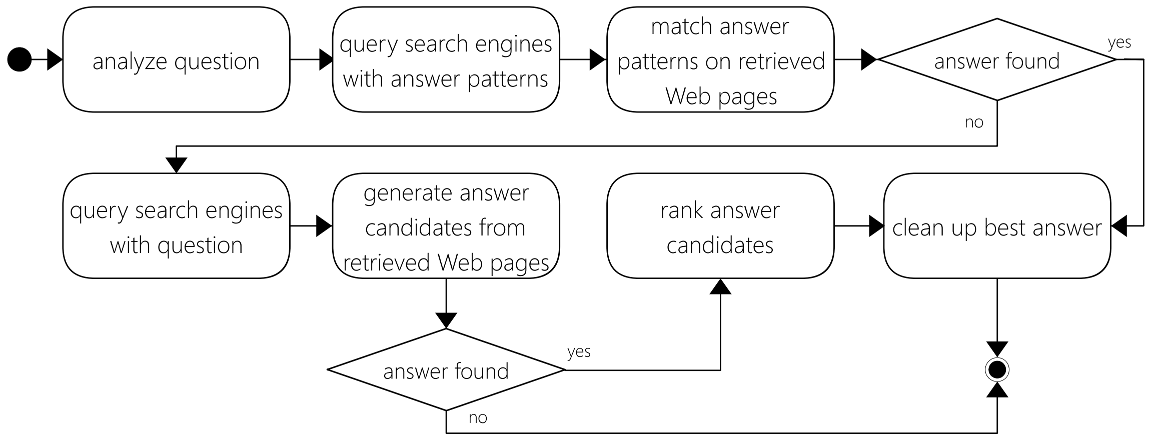

analyze question
query search engines with answer patterns
match answer patterns on retrieved Web pages
answer found
yes
no
query search engines with question
generate answer candidates from retrieved Web pages
rank answer candidates
clean up best answer
answer found
yes
no

Figure 9.6: Process of Retrieving and Extracting Answers from the Web

Analysis of the Question

Before we send the question to the search engines, we analyze the question word by word. First, a part-of-speech tagger is used to assign a POS tag to each word. Subsequently, a named entity recognizer is used to find named entities in the question. Knowing which type the entities in the question are can help us to better understand the semantics of the question and rank the answer candidates later on. For example, to answer the question "Who is more popular than Jim Carrey?", it is helpful to know that *Jim Carrey* is an entity of the concept *Actor* because we then know that the sought-after person is likely also an entity from the concept *Actor*. Later we will be able to apply NER on the answer candidates to find other actors. We also remove stop words in this step.

Answer Patterns

An "answer pattern" is a reformulation of the question so that it states the answer. For example, the question "Where was Jim Carrey born?" can be reformulated to "Jim Carrey was born in `ANSWER`", where `ANSWER` is the location that we want to know. One question can also be reformulated to multiple answer patterns. The assumption behind question reformulation is that answers that conform with the answer pattern contain precise answers. Also, the Web is a large corpus of text, requiring us to formulate precise queries to limit the number of matching search results.

Creating an answer pattern is a difficult task because there are many ways to formulate the same question, and there are no general rules on how to phrase a question as an answer. Nonetheless, for many simple questions, we can find certain patterns that repeat frequently. We analyzed hundreds of questions from the TREC dataset, WikiAnswers, True Knowledge, and Yahoo! Answers to find regularities in the questions and their answer patterns.

We identified five part-of-speech tag groups that are necessary to rephrase a question. These five groups are:

1. **Question** words, such as "What", "Who", or "Where"
2. **Auxiliary** verbs, such as "is" or "did"
3. **Verbs**, such as "jump" or "run"
4. **Subjects**, such as the (proper) nouns "Jim Carrey" or "castle"
5. **Other**, all text that does not belong in one of the other groups

Figure 9.7 provides a simple example of a question and one answer pattern for that question. We manually created 27 of these patterns to capture questions starting with "Who", "What", "When", "Where", "Why", and "How".

The answer pattern contains a placeholder for the answer (`ANSWER` in Figure 9.7). The answer placeholder can be anywhere in the pattern. The answer pattern has two purposes. First, we build a phrased search engine query to retrieve only Web pages containing the exact answer

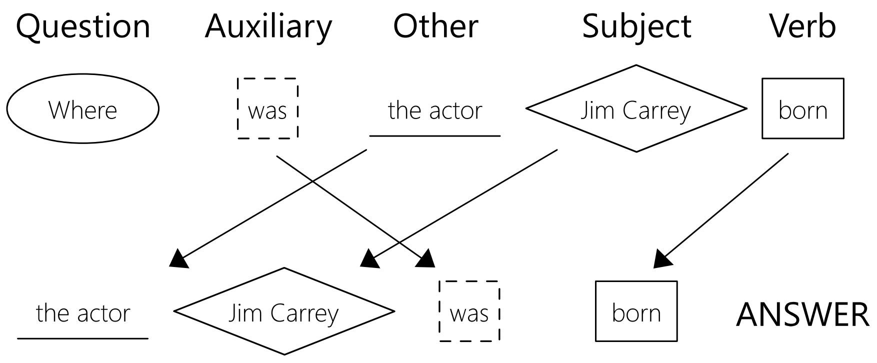


Figure 9.7: Example of an Answer Pattern for a Question

pattern, and second, we create a regular expression to search for the pattern on the text of the retrieved Web pages.

One interesting alternative to manually mapping question patterns to answer patterns is to apply supervised machine learning to this task. Creating a dataset of training data is, however, beyond the scope of this work.

Retrieving Result Pages

As our experiment with 87 test questions demonstrated, we were able to find correct answers within the top 10 results from a search engine for almost all questions. In this step, one or more search engines are queried with the question and the top result pages are analyzed. For each of the retrieved pages, the main text content is extracted. We have already shown in Section 5.2.2, that the navigation, header, and footer of a Web page contain useless content. We therefore need to find the elements of the page that contain most of the textual content that is likely to contain the answer we are looking for. We use a rule-based content extraction system for this purpose (Urbansky et al., 2011a).

Generating Answer Candidates

After extracting the main textual content of each retrieved Web page, we now create answer candidates from these pages. Every page can contain zero or more answer candidates. One answer candidate belongs to exactly one page. The answer candidate algorithm checks the relevance of each sentence in the text. A sentence is considered relevant if it contains a word from the question or a semantically related term. For example, if the question asked for "favorite movies", all detected movie names in the answer text are related to the question and the sentences containing them are considered relevant. The answer candidate extraction algorithm adds all relevant sentences to a candidate until two irrelevant sentences are detected in sequence. In this case, one answer candidate is complete, and the algorithm continues through the rest of the text.

After the answer candidate extraction step, we have a set of answer candidates, which now need to be ranked in order to find the best matching answer.

Ranking Answer Candidates

When we have several answer candidates, we need to decide which one is most likely to answer the user's question. We use a sliding window similarity approach to rank the answers. The assumption is that the answer will have a high similarity with the question. Therefore, we slide the question character by character over each answer candidate and calculate the similarity[8] between the question and answer. The highest sliding window similarity for each answer candidate is kept and compared to the highest sliding window similarities of the other answer candidates. The answer candidate with the highest sliding window similarity becomes our top-ranked answer.

We handle questions that ask for a number or a date differently than we handle other question intents. Knowing that the user is looking for one of these two answer types can be used to improve the answer extraction.

Questions with Date Intents If the user is looking for a date, we extract all mentions of dates in the main text areas of the retrieved Web pages. After normalizing all dates to the UTC format, we count their occurrences and create an answer using the date that occurs most often on the answer pages.

Questions with Number Intents If the user is asking a question to find a numeric value, we extract all numeric expressions from the main text area of the retrieved Web pages. For each extracted number, we calculate a score based on how similar the sentence containing the number is to the question. We then rank the extracted numbers by their number of occurrences and their calculated scores. We use the highest-ranked number to create the answer.

Cleaning Up the Answer

Before presenting the answer to the user, we clean up the text by applying simple styling rules. For example, we change "u" to "you", replace multiple question or exclamation marks "!!!!" with single ones "!", remove questions at the beginning of the answer, and force capitalization on the first word of every sentence. We found that these styling rules are necessary since answers might be extracted from user-generated content, which is often poorly written. Applying spelling correction could be beneficial as well.

[8]Experimentally, we found Smith-Waterman (Waterman and Smith, 1981) to be a good similarity metric to use for our purpose.

9.4 Evaluation

This section evaluates 13 different question answering approaches. To the best of our knowledge, this is the most comprehensive evaluation combining research and commercial approaches on a large set of 600 questions. Additionally we evaluate selected parts of the introduced question answering components of the WebKnox system.

9.4.1 Dataset

Our dataset[9] consists of 600 questions drawn equally from the following six sources:

1. **QALD-2 (QA)**: We took the 100 training questions from the 2nd Open Challenge on Question Answering over Linked Data (QALD-2)[10].
2. **Search Engine (SE)**: Bloor (2011) compiled a list of the most frequently asked questions on Google in 2011. We added 100 of these questions to our dataset.
3. **TREC-9 (TR)**: We used the first 100 questions from the TREC-9 dataset[11].
4. **YahooAnswers (YA)**: We took 100 of the 1,000 most recent questions on the Yahoo! Answers website[12].
5. **WikiAnswers (WA)**: We selected 100 random questions from WikiAnswers using their random question function[13].
6. **True Knowledge (TK)**: We crawled a set of approximately 11,000 questions from the True Knowledge website[14] on 11th of February 2012. We then randomly sampled 100 questions from this set.

Figure 9.8 shows the question intent distribution of the 100 questions from each of the six datasets.

We can clearly see that the question intents are different in each dataset. Questions on WikiAnswers and YahooAnswers, for example, are often questions that ask for opinions or explanations, rather than factual information. Most of the questions on Google are about procedural information or explanations. On True Knowledge, people generally ask questions about entities (Subject, Location, Thing) or let the system define a term. Presumably, people feel that the system does not provide good answers to opinion questions and hence they do not ask the system these questions. Interestingly, the QALD-2 dataset is the only dataset that contains questions about lists of things. The questions in the QALD-2 dataset are supposed

[9]We made the dataset publicly available under `http://areca.co/17/600-random-questions-from-six-datasets`, last accessed on 19th of March 2012.

[10]`http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2`, last accessed on 16th of March 2012

[11]`http://trec.nist.gov/data/topics_eng/qa_questions_201-893`, last accessed on 16th of March 2012

[12]`http://answers.yahoo.com/`, last accessed on 16th of March 2012

[13]`http://wiki.answers.com/Q/Special:Randompage`, last accessed on 16th of March 2012

[14]`http://www.trueknowledge.com/new-questions/`, last accessed on 16th of March 2012

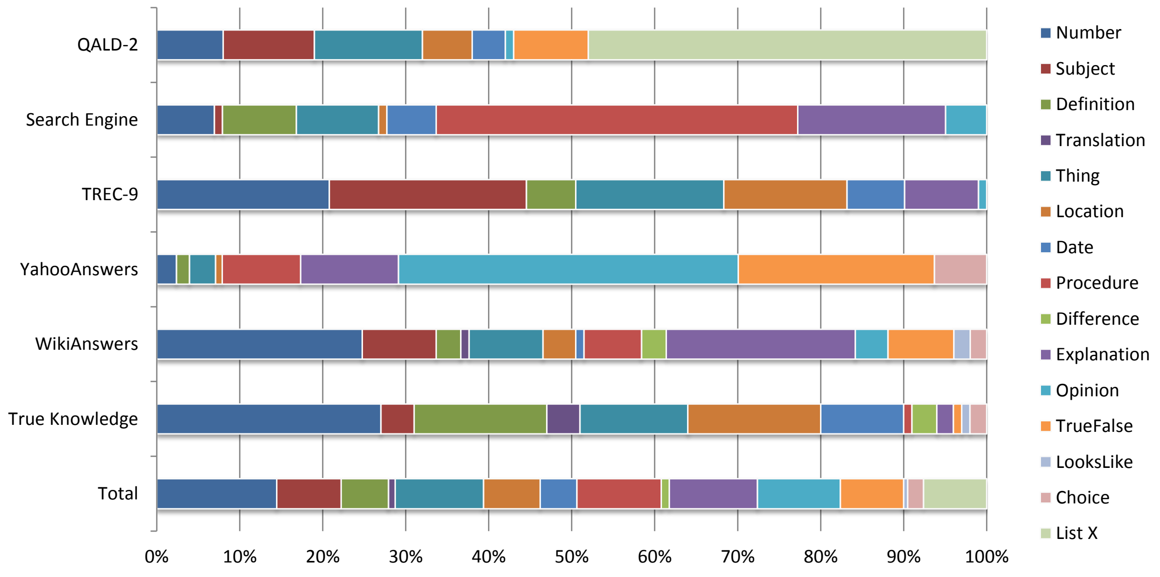


Figure 9.8: Distribution of Question Intents in the Six Datasets

to be answered using Linked Data where lists can be generated automatically. The reality, however, is that these types of questions are – at least at the moment – not frequently asked by people on the Web.

9.4.2 Focused Crawler Efficiency

In this section, we evaluate the efficiency of our focused crawling approach by comparing the number of extracted questions and the size of the URL stack to the trivial crawling approach. Since we use XPaths to find questions and answers on Web pages that have the same layout, we do not evaluate the precision of the extractions as it has to be 100 % under the given assumptions.

We use seven QA-rich websites for this experiment – WikiAnswers[15], Yahoo! Answers, Stack-Overflow[16], ChaCha[17], Mahalo Answers[18], Nobosh Answers[19], and HubPages[20]. We add up the number of extracted QAs over all websites. As a baseline, we use the trivial approach, which does not perform any classification and simply takes the next available URL from the stack. In Figure 9.9, we can see that after 5,000 URLs have been visited, WebKnox was able to extract almost twice as many questions using focused crawling compared to the baseline crawling approach. Since WebKnox prefers URLs with QAs, the stack size increases about 30 % slower and thus saves resources. Also, URLs with red prefixes are deleted from the stack, which explains the slight drops of the URL stack size in Figure 9.9 (red line) at about 300 and 2,000 visited URLs.

[15]`http://Wiki.Answers.com`, last accessed on 20th of June 2012
[16]`http://StackOverflow.com`, last accessed on 20th of June 2012
[17]`http://ChaCha.com`, last accessed on 20th of June 2012
[18]`http://Mahalo.com`, last accessed on 20th of June 2012
[19]`http://Answers.Nobosh.com`, last accessed on 20th of June 2012
[20]`http://HubPages.com/Answers/latest`, last accessed on 20th of June 2012

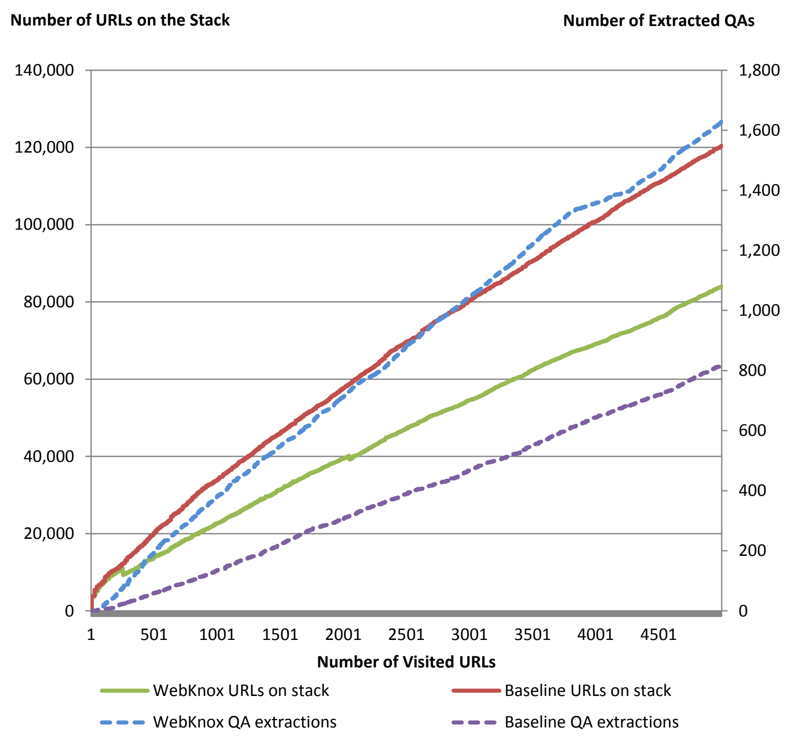


Figure 9.9: Comparison of Efficiency between a Non-focused Baseline Crawler and the Focused Crawler

9.4.3 Question Intent Classification

Our rule-based question intent classifier classifies 53 % of the questions in the dataset correctly. Our question answering approach that extracts answers from the Web is the only approach that makes use of the classified question intent. This approach makes a distinction between the date and number intents of questions while treating all other intents (see Section 9.2) identically. It is therefore important that the detection of the date and number intents performs well. The classification precision of the date intent is about 87 % at a recall of about 71 %. The precision for the number intent is approximately 92 % at a recall of about 65 %.

The precision is more important than the recall since false negative classifications are treated the same as other questions and could still result in correct answers. False positive classifications, however, would most likely lead to wrong answers since only dates or numbers are considered possible answers, and in these cases no date or number is expected.

9.4.4 Comparison of Question Answering Systems

We now compare the performances of the 13 question answering approaches on each of the six datasets and on each of the 15 question intents.

Since not all questions have only one correct answer, we could not evaluate the systems automatically. Instead, we decided to conduct a user study. We let three raters judge each answer in terms of relevance and rank. The relevance value for an answer could have three values: “x” when there was no answer, zero if the answer was not relevant to the question, and one if the rater decided that the answer was related to the question (regardless of whether it was correct or not). If an answer was judged relevant, the rater then had to decide about the quality of the answer relative to the other relevant answers from other systems. For this purpose, the rater assigned a rank value between 1 (best answer) to 13 (worst answer) to each relevant answer. If two answers were the same quality, they could also get the same rank. If only one answer was correct and the system did not find the correct answer, the rank of the system is “x”.

We calculated the Fleiss Kappa (Fleiss, 1981) value between the three raters to be 0.63, which is “substantial agreement” according to Landis and Koch (1977).

A “true positive” for the precision, recall, and F1 calculation is an answer that was judged relevant by the raters. Relevant answers can still be incorrect for factual questions so we capture these cases with an “Error Rate”, which is the ratio of incorrect answers to relevant answers.

We evaluated the three question answering approaches of WebKnox against all the other approaches. The matching approach is denoted with “WX: Match”, the computation approach is labeled “WX: Compute” and additionally, the approach that extracts answers from the Web is divided into “WX: Extraction Ref” (using reformulation and answer patterns) and “WX: Extraction Sim” (utilizing the similarity of answer candidates to the question). For the “WX: Match” approach, we used the focused crawler (see Section 9.3.1) to create a database of 293,089 questions and 1,540,174 answers. For the question matching, we used simple natural language matching provided by the database.

Comparison by Dataset

We first compare the precision and recall of the 13 question answer approaches across all intents in each of the six datasets. Table 9.2 shows the precision (first lines) and recall (second lines) values of the systems for each dataset.

We can see that whenever Google gave a direct answer, it was always relevant. Google did not answer any questions from the YahooAnswers and WikiAnswers datasets in which many questions require a rather complicated explanation. In general, Google is very selective about which queries to answer, yielding a low recall of only 3 %. Groovy Answers’ strength in answering factual questions over RDF is seen in its high precision and the highest recall for questions from the QALD-2 dataset.

AQUA was built to answer opinion-related questions especially well and shows a comparatively high recall for the Search Engine, YahooAnswers, and WikiAnswers datasets, which contain many of these question intents.

OpenEphyra brings up relevant answers for more than half of the questions from the TREC-9 dataset – more than any other system – but fails for questions from other corpora such as YahooAnswers. The highest precision in the TREC-9 dataset is again reached by Google.

Unsurprisingly, True Knowledge retrieves 81 % relevant answers with 99 % precision to questions from the True Knowledge dataset. Google answers only 2 % of the questions but all answers are relevant. More interestingly, however, True Knowledge – a system built primarily on a knowledge base – was able to produce between 93 % and 100 % relevant answers to questions from the datasets YahooAnswers and WikiAnswers, even though they often require more sophisticated answers. Even the Yahoo! Answers system was not able to answer questions taken from its own platform with a precision higher than 84 %. Unsurprisingly, 70 % of the questions from the YahooAnswers dataset could be answered using Yahoo! Answers itself. However, Yahoo! Answers did not perform well on questions from WikiAnswers, which are generally rather similar. The Web extraction approach of WebKnox yields the highest recall for the WikiAnswers dataset.

On average, Google answers questions with 100 % precision but the WebKnox Web extraction approach yields the highest recall with 42 % relevant answers over all six datasets.

Against our expectations, WX: Compute achieved a lower precision than the Web extraction-based approaches of WebKnox. We think this is due to the rather simple answer computing approach. A similar approach to WX: Compute is Groovy Answering which yields more than twice the precision. Moreover, the computing approach is rather selective and answers were found for only few questions which gives us a low confidence in the precision and recall values.

With 35 % precision and 14 % recall on average, the questions from the YahooAnswers dataset were the most difficult for the systems to answer. Scores for the WikiAnswers dataset are only slightly better. We can therefore conclude that current approaches still have problems answering questions that need a long, narrative answer.

Dataset	**True Knowledge**	**Groovy Answers**	**Wolfram|Alpha**	**Google**	**START**	**Yahoo! Answers**	**AQUA**	**OpenEphyra**	**Power Aqua**	**WX: Match**	**WX: Compute**	**WX: Extract Ref**	**WX: Extract Sim**	**Avg**
QA	0.96	0.80	0.82	**1.00**	0.68	0.19	0.14	0.49	0.21	0.10	0.55	0.67	0.32	0.53
	0.32	**0.43**	0.14	0.06	0.18	0.12	0.14	0.22	0.15	0.10	0.17	0.15	0.32	0.19
SE	0.65	0.47	0.67	**1.00**	0.58	0.48	0.60	0.23	0.03	0.33	0.00	0.49	0.56	0.47
	0.18	0.12	0.08	0.01	0.20	0.37	**0.58**	0.16	0.01	0.32	0.00	0.30	0.54	0.22
TR	0.94	0.64	0.85	**1.00**	0.83	0.43	0.34	0.56	0.16	0.20	0.25	0.59	0.51	0.56
	0.35	0.28	0.21	0.09	0.38	0.35	0.32	**0.52**	0.09	0.20	0.03	0.35	0.51	**0.28**
YA	**1.00**	0.13	0.00	-	0.67	0.84	0.51	0.02	0.04	0.22	0.00	0.37	0.44	0.35
	0.01	0.01	0.00	0.00	0.02	**0.70**	0.44	0.00	0.01	0.22	0.00	0.03	0.42	0.14
WA	**0.93**	0.49	0.83	-	0.79	0.31	0.37	0.39	0.03	0.20	0.04	0.27	0.41	0.42
	0.05	0.06	0.07	0.00	0.06	0.13	0.33	0.18	0.01	0.19	0.00	0.08	**0.39**	0.12
TK	0.99	0.78	0.93	**1.00**	0.85	0.38	0.40	0.60	0.11	0.20	0.50	0.59	0.42	**0.60**
	0.81	0.27	0.26	0.02	0.26	0.13	0.23	0.32	0.06	0.19	0.01	0.21	0.34	0.24
Avg	0.91	0.55	0.68	**1.00**	0.73	0.44	0.39	0.38	0.10	0.21	0.22	0.50	0.44	
	0.28	0.20	0.13	0.03	0.19	0.30	0.34	0.23	0.06	0.21	0.04	0.19	**0.42**	

Table 9.2: Precision and Recall of the Question Answering Approaches by Dataset

Comparison by Intent

We now compare the precision and recall of the 13 question answering approaches across all datasets in each of the 15 question intents. Table 9.3 shows the precision (first lines) and recall (second lines) values of the systems for each question intent.

To our surprise, the Web extraction-based system OpenEphyra scored the highest recall on questions with numeric intents with 55 %. We expected a computation-based approach to score highest, but True Knowledge has only the third best recall for this intent. We make a similar observation for questions that expect a date as an answer.

The table supports the thesis that Google only answers factual questions. Again, Google's answer precision is 100 % for the intents it answers. Definitions are answered by Google too, but not in the same way as other questions, hence it does not show up in the table. WebKnox's computation approach was able to answer questions with a definition intent most precisely.

Only TrueKnowledge, Wolfram | Alpha, and Power Aqua seem to have appropriate support for questions that ask for translations, reaching a maximum precision of 78 %.

Procedure questions are answered equally well with 53 % precision by Yahoo! Answers and AQUA, while other systems with similar approaches, such as OpenEphyra, seem to fail completely. As expected, systems following the computing approach to question answering (for example, Groovy Answers, Power Aqua, and WX: Compute) fail in answering procedural questions as well.

Questions asking about the difference between things can be answered with a precision of up to 100 % by True Knowledge and START. True Knowledge can also answer explanation intent questions with a precision of 97 %. This result is interesting because we expected explanation intent questions to be as difficult as procedural questions, but on average they are answered 20 % more precisely.

Also to our surprise, True Knowledge was able to come up with 100 % relevant answers for opinion-related questions. We expected the matching and extraction-based systems to achieve a higher precision for this intent. As expected, however, opinion-related questions cannot be answered with a high recall by computation-based approaches. For this intent, only systems designed to match questions against a knowledge base or extract opinions related to the question from the Web succeed. Yahoo! Answers, AQUA, and WX: Extract show the highest recalls for opinion intents.

TrueFalse and Choice questions were answered with 100 % relevant answers by True Knowledge. LooksLike and List X questions could be answered with 100 % precision by START, WX: Compute, and Google.

Questions that ask for lists of things were only present in the QALD-2 dataset, which can be especially well answered by Groovy Answers. Groovy Answers only reaches a recall of 36 %, but that is almost twice the recall from the second best system – True Knowledge – for this intent.

On average, Google is the system with the most precise answers, mostly because it leaves out many complex questions. Although Google and True Knowledge are the most precise systems, other systems can answer some question intents more precisely. If one could classify the question intent accurately, a combination of these systems could increase the average answering precision further.

The best recall scores are reached primarily by systems that use the Web extraction approach, such as AQUA, OpenEphyra, and WX: Extract Sim. WebKnox's Web extraction approach scores the highest recall with 41 % on average over all intents, while Google – the system with the most precise answers – exhibits the lowest recall of only 2 %.

Questions regarding definitions can be answered with an average precision of 61 % and an average recall of 32 %, making them the easiest questions across all systems. Questions with a procedural and a TrueFalse intent are the most difficult, and can only be answered with a precision of 25 % and a recall of 10 % respectively on average over all 13 systems.

Overall Comparison

We have compared the precision and recall for all 13 approaches across the six datasets and all 15 question intents. We have to combine this information to see which system is best overall. Table 9.4 shows the overall performance of the systems. "Rank" is the average rank that the raters have assigned to a relevant (and correct) answer. The lower the average rank, the better the average quality of relevant answers. "OAR" stands for Only Answer Resource and shows what percentage of questions could **only** be answered by that one system. The "Error Rate" is the percentage of answers that were rated relevant to the factual questions but were definitely wrong.

Yahoo! Answers has the best rank on average, which is not surprising since human users write better quality answers than machines can compute or extract from websites. Yahoo! Answers is also the system that has the highest OAR, that is, for 2.08 % of the questions, only this system was able to produce a relevant answer.

Google has the highest answering precision, but also the lowest recall of all systems. Additionally, Google's error rate of only 1.85 % emphasizes its focus on answering questions correctly and precisely, at the expense of answering far fewer questions. All the relevant answers Google produced could also be produced by one of the other systems, hence the OAR of 0 %.

The Web extraction approach of WebKnox (WX: Extract Sim) yields the highest recall of relevant answers with 42 % and shares the highest F1 value of 43 % with True Knowledge. The error rate of True Knowlege is only 4.10 % – much lower than WebKnox's 19.68 % error rate. True Knowledge's rank is also 0.36 lower than WebKnox's rank. Thus, True Knowledge is the overall best question answering system in this evaluation.

Figure 9.10 visually compares the precision, recall, and F1 values for each system on average over all datasets used.

Thesis 5 (see Section 1.4) stated that ontology-based question answering systems that compute answers are more precise at the cost of a lower recall. Table 9.5 shows the average precision, recall, and F1 values of the 13 evaluated systems grouped by their approach (see Table 9.1 for information about which systems belong in which groups). The data confirms our hypothesis – computing approaches are the most precise with 58 % on average, but also yield the lowest recall of only 12 %. The best approach is the extraction from the Web with an average F1

Intent	**True Knowledge**	**Groovy Answers**	**Wolfram|Alpha**	**Google**	**START**	**Yahoo! Answers**	**AQUA**	**OpenEphyra**	**Power Aqua**	**WX: Match**	**WX: Compute**	**WX: Extract Ref**	**WX: Extract Sim**	**Avg**
Number	0.97	0.42	0.94	**1.00**	0.89	0.36	0.27	0.65	0.01	0.19	0.08	0.39	0.52	0.51
	0.38	0.16	0.31	0.03	0.23	0.16	0.23	**0.55**	0.01	0.19	0.01	0.16	0.47	0.22
Subject	0.94	0.89	0.76	**1.00**	0.73	0.47	0.33	0.35	0.27	0.22	0.52	0.42	0.53	0.57
	0.33	0.33	0.11	0.08	0.16	0.38	0.30	0.25	0.18	0.22	0.07	0.19	**0.52**	0.24
Def.	0.93	0.97	0.76	-	0.78	0.52	0.56	0.31	0.17	0.26	**1.00**	0.59	0.54	**0.61**
	0.75	0.68	0.15	0.00	0.65	0.31	0.30	0.20	0.05	0.24	0.03	0.34	0.42	**0.32**
Transl	**0.78**	-	0.67	-	-	0.33	-	-	0.67	0.07	-	-	0.27	0.46
	0.47	0.00	0.13	0.00	0.00	0.07	0.00	0.00	0.40	0.07	0.00	0.00	0.27	0.11
Thing	0.94	0.74	0.97	**1.00**	0.82	0.48	0.40	0.43	0.08	0.27	0.33	0.63	0.42	0.58
	0.34	0.20	0.16	0.08	0.19	0.33	0.36	0.25	0.04	0.27	0.05	0.24	**0.41**	0.22
Loc.	0.97	0.64	0.83	**1.00**	0.87	0.37	0.26	0.65	0.29	0.20	0.33	0.84	0.35	0.58
	0.47	0.33	0.12	0.07	0.36	0.16	0.22	**0.47**	0.17	0.19	0.06	0.41	0.34	0.26
Date	**1.00**	0.53	0.94	**1.00**	0.83	0.29	0.27	0.90	0.08	0.05	0.08	0.42	0.60	0.54
	0.61	0.23	0.37	0.07	0.18	0.17	0.20	**0.64**	0.04	0.05	0.01	0.29	0.54	0.26
Proc.	0.13	0.00	-	-	0.27	**0.53**	**0.53**	0.01	0.00	0.33	0.00	0.46	0.45	0.25
	0.02	0.00	0.00	0.00	0.06	0.41	**0.52**	0.01	0.00	0.33	0.00	0.21	0.43	0.15
Diff.	**1.00**	-	0.33	-	**1.00**	0.50	0.47	0.00	0.00	0.11	0.00	-	0.33	0.37
	0.50	0.00	0.11	0.00	0.17	0.17	0.39	0.00	0.00	0.11	0.00	0.00	0.28	0.13
Expl.	**0.97**	0.64	0.42	-	0.69	0.49	0.59	0.27	0.03	0.24	0.00	0.55	0.52	0.45
	0.14	0.14	0.02	0.00	0.17	0.31	**0.58**	0.12	0.01	0.24	0.00	0.24	0.50	0.19
Opinion	**1.00**	0.17	-	-	0.00	0.80	0.54	0.10	0.03	0.22	0.00	0.33	0.49	0.34
	0.02	0.02	0.00	0.00	0.00	**0.63**	0.48	0.02	0.01	0.22	0.00	0.04	0.49	0.15
TrueF.	**1.00**	-	0.33	-	0.78	0.54	0.25	0.00	0.08	0.11	0.33	-	0.28	0.37
	0.16	0.00	0.03	0.00	0.08	**0.33**	0.23	0.00	0.03	0.11	0.03	0.00	0.28	0.10
LooksL.	-	-	0.00	-	**1.00**	0.11	0.56	0.00	0.00	0.44	**1.00**	0.33	0.67	0.41
	0.00	0.00	0.00	0.00	0.33	0.11	0.56	0.00	0.00	0.44	0.11	0.33	**0.67**	0.20
Choice	**1.00**	-	-	-	-	0.50	0.67	-	0.00	0.08	-	-	0.33	0.43
	0.25	0.00	0.00	0.00	0.00	**0.50**	**0.50**	0.00	0.00	0.08	0.00	0.00	0.33	0.13
List X	0.97	0.79	0.88	**1.00**	0.75	0.09	0.11	0.58	0.21	0.06	0.69	0.83	0.16	0.55
	0.20	**0.36**	0.15	0.02	0.13	0.05	0.10	0.13	0.16	0.06	0.17	0.03	0.16	0.13
Avg	0.90	0.58	0.65	**1.00**	0.72	0.43	0.42	0.33	0.13	0.19	0.34	0.53	0.43	
	0.31	0.16	0.11	0.02	0.18	0.27	0.33	0.18	0.07	0.19	0.04	0.17	**0.41**	

Table 9.3: Precision and Recall of the Question Answering Approaches by Intent

value of 33 %, but since this score is still unsatisfactory, a combination of different approaches should be favored. True Knowledge already combines approaches by presenting answers from the kgb Answers database when it is unable to compute an answer.

System	Rank	OAR	Error Rate	Precision	Recall	F1
True Knowledge	**1.21**	1.28 %	4.10 %	0.91	0.28	**0.43**
Groovy Answers	1.44	0.28 %	5.98 %	0.55	0.20	0.29
Wolfram | Alpha	1.35	0.19 %	5.70 %	0.68	0.13	0.21
Google	1.32	0.00 %	**1.85 %**	**1.00**	0.03	0.06
START	1.38	0.06 %	3.88 %	0.73	0.19	0.30
Yahoo! Answers	1.27	**2.08 %**	9.11 %	0.44	0.30	0.36
AQUA	1.40	1.69 %	10.02 %	0.39	0.34	0.36
OpenEphyra	1.82	0.25 %	33.57 %	0.38	0.23	0.29
Power Aqua	1.82	0.39 %	69.90 %	0.10	0.06	0.07
WX: Match	1.64	0.47 %	20.00 %	0.21	0.21	0.21
WX: Compute	1.60	0.06 %	10.94 %	0.22	0.04	0.06
WX: Extract Ref	1.56	0.39 %	8.96 %	0.50	0.19	0.27
WX: Extract Sim	1.57	1.94 %	19.68 %	0.44	**0.42**	**0.43**

Table 9.4: Overall Comparison of the Question Answering Systems

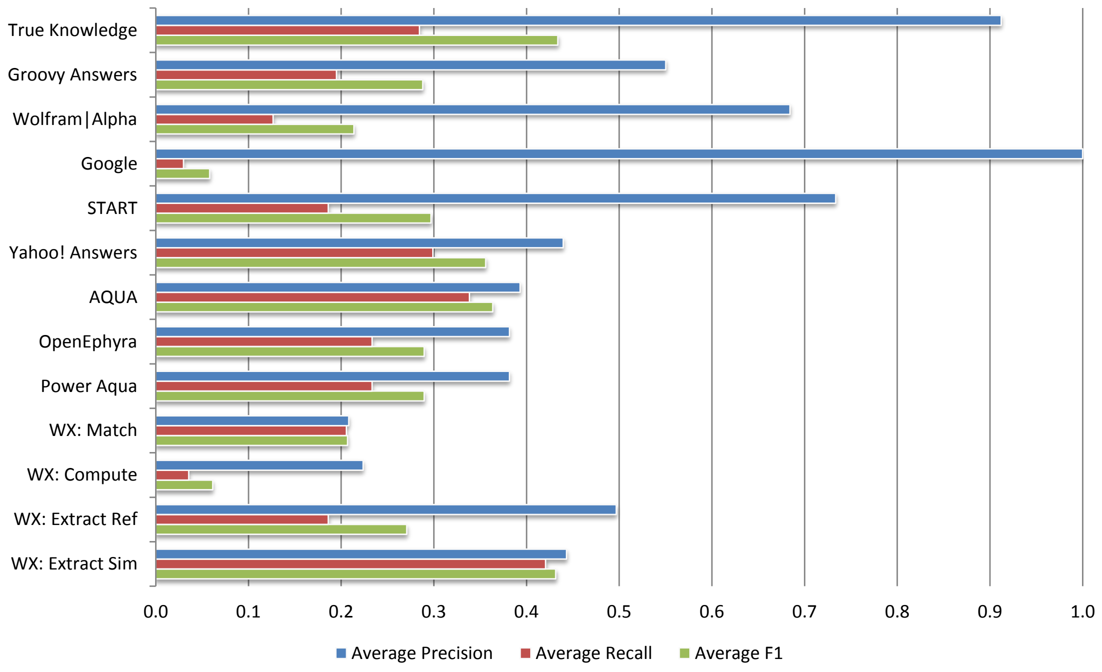


Figure 9.10: Overall Comparison of the Question Answering Systems

9.5 Summary

This chapter reviewed and compared state-of-the-art question answering systems. First, we gave an overview of related work on systems and approaches for answering natural language

Approach	**Precision**	**Recall**	**F1**
Matching	0.32	0.25	0.28
Computing	**0.58**	0.12	0.19
Extracting	0.49	**0.27**	**0.33**

Table 9.5: Comparison of Question Answering Approaches

questions and query classification. We classified question answering systems into three groups: **matching**, **computing**, and **extracting**. We then created three WebKnox question answering techniques – one for each of the approaches. For a fair comparison of the 13 question answering systems, we created a set of 600 questions from six different datasets. After reading hundreds of real user questions, we also identified 17 question intents and classified all questions in the dataset manually. A group of raters manually evaluated the answers of the systems allowing us to compare all systems across the six datasets and the question intents. Our hypothesis stating that systems that compute an answer are most precise with a low recall was confirmed by the evaluation.

There are interesting directions for future work that are out of the scope of this thesis. We have seen that different approaches excel at different question intents; a combination of these systems could create a meta question answering system that is more precise than any single system. We have presented a rule-based answer pattern generation algorithm to support extraction of answers from the Web. It would be interesting to find out whether answer patterns can also be automatically learned using supervised machine learning. Furthermore, generating texts for computed answers from a knowledge base is a topic that has much potential. Furthermore, Hensel (2011) has presented a pattern extraction approach using the WebKnox knowledge base showing how simple answer sentences can be created. The quality of the generated sentences was already comparable to Wikipedia texts.

Chapter 10

Applications

In this chapter, we show how WebKnox components and the knowledge base that we create with WebKnox can be used in practical applications. Before we present practical applications, we briefly give implementation details about which languages were used and which other libraries and toolkits were employed to create WebKnox.

The core WebKnox project is written in Java 7 and built with the project management tool Maven[1]. The entire project consists of about 25,000 lines of code in over 330 classes. The most important libraries WebKnox depends on are:

- Jena (Carroll et al., 2004) and Sesame (Broekstra and Kampman, 2002) for storing and querying ontological information
- Apache Commons[2] for common programming tasks that are not natively supported by the Java programming language
- HTML Unit[3] and NekoHTML[4] for parsing HTML pages
- Palladian (Urbansky et al., 2011a) for a collection of named entity recognizers, part-of-speech taggers, Web page retrievers, and classification approaches

10.1 Ontology Engineering with Ontofly

Ontofly is a Web-based ontology engineering tool devloped by Willner (2011). In Section 7.1, we showed how WebKnox uses this tool to create an ontology to guide the fact extraction process. The dependency between the two projects is bidirectional, since Ontofly uses WebKnox to detect fact candidates on Web pages. Figure 10.1 depicts the architecture of Ontofly and shows how it is connected to WebKnox. The user browses online on the Ontofly Web application. Requests are sent to the Jersey API which communicates with Ontofly. Ontofly serves as a component between the user and WebKnox and its ontology.

[1]`http://maven.apache.org/`, last accessed on 19th of March 2012
[2]`http://commons.apache.org/`, last accessed on 19th of March 2012
[3]`http://htmlunit.sourceforge.net/`, last accessed on 19th of March 2012
[4]`http://nekohtml.sourceforge.net/`, last accessed on 19th of March 2012

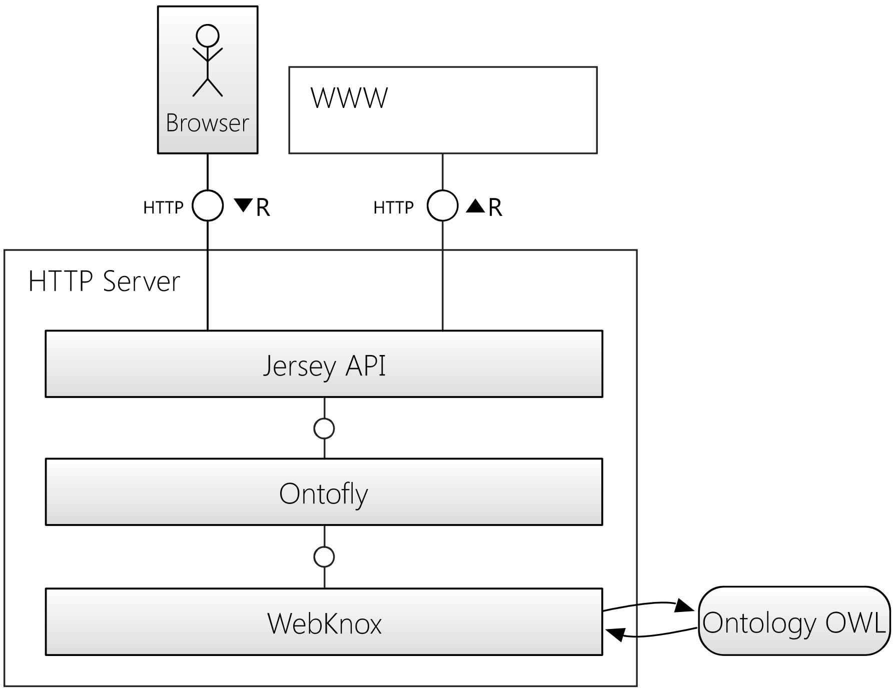
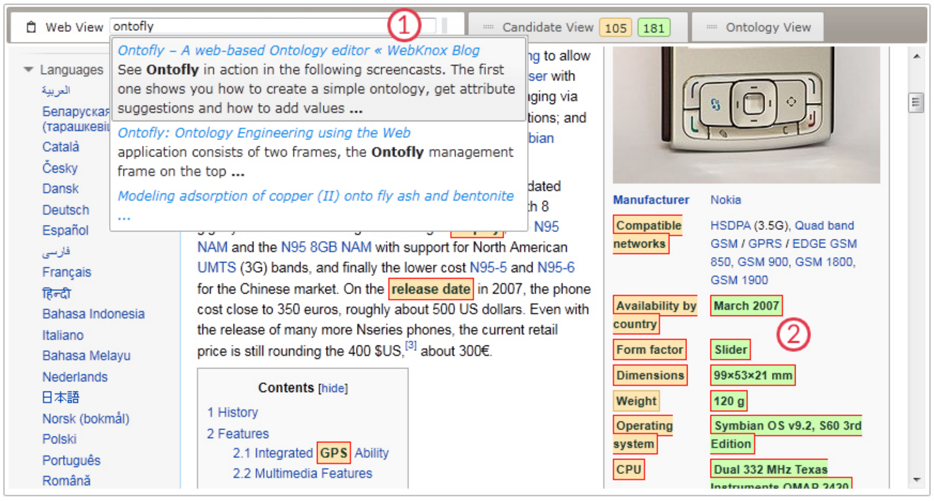


Figure 10.1: Architecture of Ontofly and Connection to WebKnox

Figure 10.2 shows a screenshot of Ontofly's Web view. At the top, the user types a term into the search bar and visits a Web page. On this Web page, WebKnox detects fact candidates. A fact consists of an attribute (marked orange) and a value (marked green). All attributes and values that have not yet been stored in the ontology are framed with a red border. The user can now quickly see attributes that might match the concept he is working on.



Ontofly – A web-based Ontology editor « WebKnox Blog
See **Ontofly** in action in the following screencasts. The first one shows you how to create a simple ontology, get attribute suggestions and how to add values **...**

Ontofly: Ontology Engineering using the Web
application consists of two frames, the **Ontofly** management frame on the top **...**

Modeling adsorption of copper (II) onto fly ash and bentonite
...

NAM and the N95 8GB NAM with support for North American UMTS (3G) bands, and finally the lower cost N95-5 and N95-6 for the Chinese market. On the release date in 2007, the phone cost close to 350 euros, roughly about 500 US dollars. Even with the release of many more Nseries phones, the current retail price is still rounding the 400 $US,[3] about 300€.

Contents [hide]
1 History
2 Features
2.1 Integrated GPS Ability
2.2 Multimedia Features

Manufacturer	Nokia
Compatible networks	HSDPA (3.5G), Quad band GSM / GPRS / EDGE GSM 850, GSM 900, GSM 1800, GSM 1900
Availability by country	March 2007
Form factor	Slider
Dimensions	99×53×21 mm
Weight	120 g
Operating system	Symbian OS v9.2, S60 3rd Edition
CPU	Dual 332 MHz Texas Instruments OMAP 2420

Figure 10.2: Screenshot of the Web-based Ontology Engineering Tool Ontofly

10.2 Question Answering Portal

We devoloped an online search service to allow users to access information from the WebKnox knowledge base and to ask questions. The website `webknox.com` is the frontend to all the information that we extracted from the Web with the techniques that we described in the previous chapters. In our motivation, we argued that question answering can improve user satisfaction compared to standard search, where the user enters a query and has to click through result documents to find what he needs. The goal of the website is to answer questions as **quickly**, **directly**, and **completely** as possible.

Figure 10.3 depicts the deployment architecture of the WebKnox system. We have two servers, each connected to a different database. The Web server hosts the Web application and the application server runs the WebKnox extraction services. The two server architecture is due to the resource requirements of the extraction services. New extractions are written to the core database, which is copied to the Web server once a day.

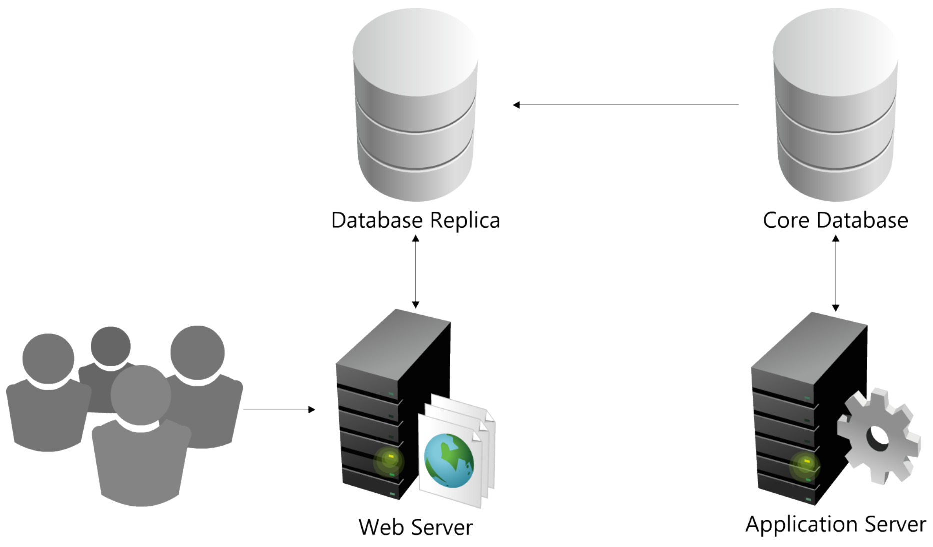


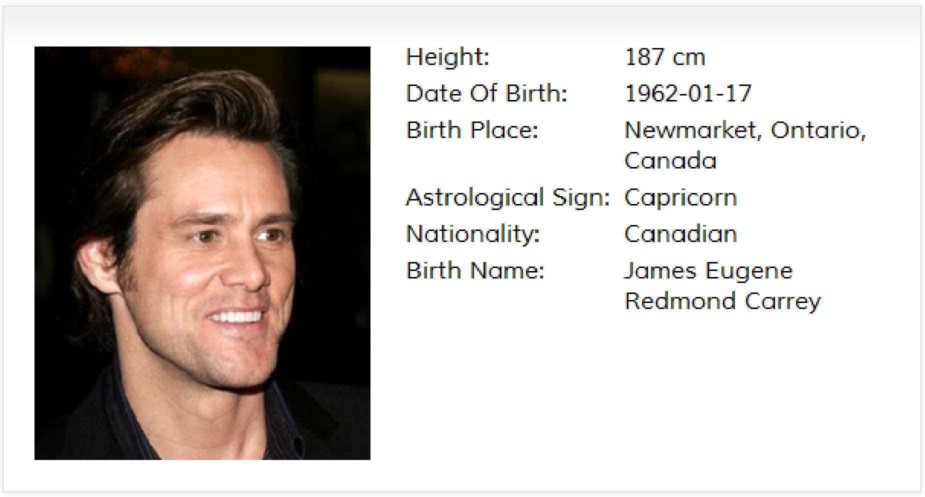
Figure 10.3: Server Deployment of the WebKnox System

The Web application is written in the scripting language PHP Hypertext Processor (PHP) and employs the Model View Controller (MVC) pattern. To fulfill our goal of making the information available **quickly**, we utilize server side caching techniques and employ several best practices for making the site load fast. These best practices include combining multiple JavaScript and style sheet files, reducing the number of HTML tags in the page, and using sprite images.

Figure 10.4 and Figure 10.5 show the Web interface of WebKnox. In Figure 10.4, the user query was just the name of an entity (*Jim Carrey*). In Figure 10.5, the user asked a non-factual question. We can see that in both cases the user gets **direct** answers to his or her questions.

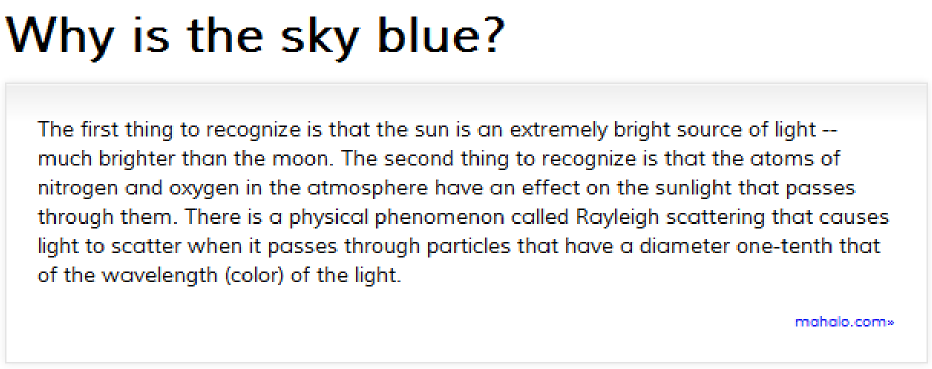
At the time of this writing, the WebKnox knowledge base[5] contains over 17 concepts, 1.3

[5]Not all the entries in the knowledge base are correct; the application filters entries with a low trust score.



Height:	187 cm
Date Of Birth:	1962-01-17
Birth Place:	Newmarket, Ontario, Canada
Astrological Sign:	Capricorn
Nationality:	Canadian
Birth Name:	James Eugene Redmond Carrey

Figure 10.4: Entity-related Information on the WebKnox Web Interface



Why is the sky blue?

The first thing to recognize is that the sun is an extremely bright source of light -- much brighter than the moon. The second thing to recognize is that the atoms of nitrogen and oxygen in the atmosphere have an effect on the sunlight that passes through them. There is a physical phenomenon called Rayleigh scattering that causes light to scatter when it passes through particles that have a diameter one-tenth that of the wavelength (color) of the light.

mahalo.com»

Figure 10.5: Question Answering on the WebKnox Web Interface

million entities, 60 thousand facts, 325 thousand questions, and 1.8 million answers. This is still far from our goal to make the knowledge base **complete**.

10.3 Application Programming Interface

We want to allow not only human beings, but also machines to access the information in the knowledge base. To accomplish this, we developed an API that allows machines to read and process information from the knowledge base in a structured manner. We decided to follow the REST (Fielding, 2000) architecture for our API and use JSON as a serialization format. In REST, every URI is a resource. Our resources are entities and questions. Information about entities and answers to questions can be obtained by sending HTTP GET requests to the following URIs[6]:

[6]Note that we use an application id and application key to authorize requests. The API can be tested online at `http://webknox.com/api`.

- `http://webknox.com/api/entities/filter?entityName=ENTITY` retrieves all entities that match the given name `ENTITY`. Since entity names can be ambiguous, the application has to decide about which entity it wants to obtain more information. Each entity in the result has an identifier and the name of the concept to which it belongs. The identifier `ENTITYID` can then be used in the next call to retrieve information about the entity.
- `http://webknox.com/api/entities/ENTITYID` retrieves information about the entity with the identifier `ENTITYID`.
- `http://webknox.com/api/questions/answers?question=QUESTION` retrieves answers for the question `QUESTION`.

Figure 10.6 shows an example JSON response to a request to the entities resource with the entity *Canon EOS 50D*.

```
[
  {
    "mentionUrls": [
      "consumersearch.com/digital-cameras/index",
      "en.wikipedia.org/wiki/List_of_cameras_supporting_a_raw_format",
      "alamy.com/contributor/help/recommended-digital-cameras.asp",
      "en.wikipedia.org/wiki/List_of_cameras_supporting_a_raw_format"
    ],
    "facts": [
      {
        "effective pixels": "10100000"
      },
      {
        "image processor": "DIGIC 4"
      },
      {
        "weight": "730"
      },
      {
        "monitor": "3 inch TFT LCD"
      }
    ],
    "name": "Canon EOS 50D",
    "confidence": 0.5
  }
]
```

Figure 10.6: Example JSON Response to a Request to the Entities Resource

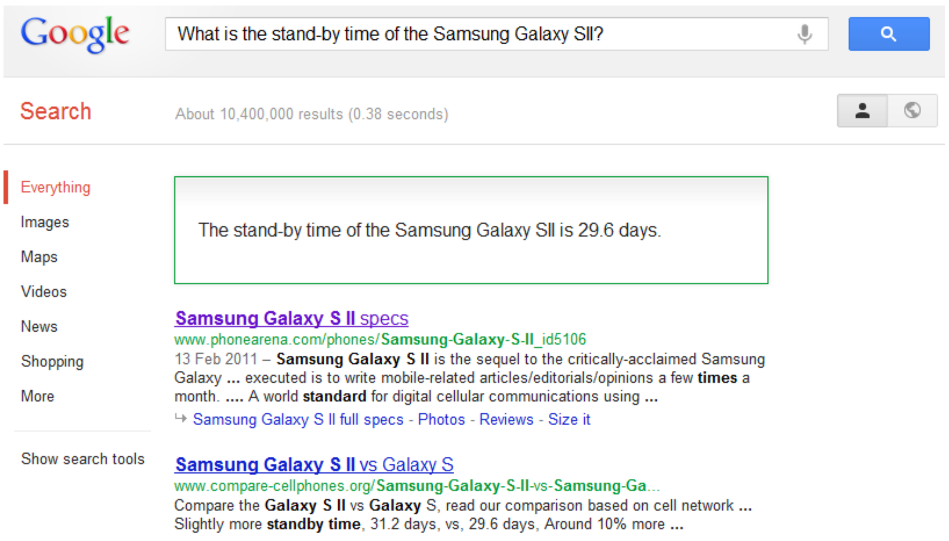
Figure 10.7 shows an example JSON response to a request to the questions resource with the question “Where was Jim Carrey born?”.

```
{
  "answer": "Jim Carrey was born in Canada. He was the younger [sic] of four
             siblings and once admitted that had he not become a comedian,
             he would have worked probably been working [sic] the steel
             mills of Ontario. Jim Carrey was born in Newmarket, Ontario,
             Canada on January 17th, 1962 [...]"
}
```

Figure 10.7: Example JSON Response to a Request to the Questions Resource

10.4 Search Engine Enhancer

Since most Web users use the same search engine on a daily basis, it is unlikely that they will visit a certain platform to search for answers. For example, many users even search on Google when they know that they want information from Wikipedia. Because of this behavior, we cannot expect users to visit the portal described in Section 10.2. We therefore built a browser extension for Chrome to enhance the search with results from WebKnox. Figure 10.8 shows the enhancer in action on the Google search result page. The box with the green border contains an answer to the Google query that comes from WebKnox. For many simple questions, users no longer need to follow links to get their answers.


Google
What is the stand-by time of the Samsung Galaxy SII?
Search
About 10,400,000 results (0.38 seconds)
Everything
Images
Maps
Videos
News
Shopping
More
Show search tools

The stand-by time of the Samsung Galaxy SII is 29.6 days.

Samsung Galaxy S II specs
www.phonearena.com/phones/**Samsung-Galaxy-S-II**_id5106
13 Feb 2011 – **Samsung Galaxy S II** is the sequel to the critically-acclaimed Samsung Galaxy ... executed is to write mobile-related articles/editorials/opinions a few **times** a month. .... A world **standard** for digital cellular communications using ...
↳ Samsung Galaxy S II full specs - Photos - Reviews - Size it

Samsung Galaxy S II vs Galaxy S
www.compare-cellphones.org/**Samsung-Galaxy-S-II**-vs-**Samsung-Ga**...
Compare the **Galaxy S II** vs **Galaxy** S, read our comparison based on cell network ...
Slightly more **standby time**, 31.2 days, vs, 29.6 days, Around 10% more ...

Figure 10.8: Example Question and WebKnox Enhancement for Google on Chrome

The enhancer extension for Chrome supports Google, Bing, Yahoo, DuckDuckGo, and Ask, which means that 93.84 % of searches on the Web[7] could be enhanced using the extension.

[7]According to `http://www.karmasnack.com/about/search-engine-market-share/` (last accessed on 19th of March 2012) the search engine market share is as follows: Google: 86.77%, Baidu: 4.46%, Yahoo: 4.40%, Bing: 2.11%, and Ask: 0.56%.

10.5 Summary

This chapter presented examples of practical applications that we developed using the knowledge base. We created proof-of-concept applications that show that machines as well as human beings can benefit from the extracted information using the API and the Web application respectively.

Chapter 11

Epilogue

To summarize this thesis, we will review the conclusions of each chapter and describe the answers to our research questions. Furthermore, we will outline the main contribution of this work and highlight interesting directions for future research.

11.1 Chapter Review

Chapter 1 motivated this thesis by presenting three use cases in which a knowledge base is beneficial. We paid particular attention to the question answering use case throughout the thesis and devoted an entire chapter to this subject. After stating the requirements that the desired knowledge base should fulfill, we set the focus and limitations for the scope of this thesis. We finished the first chapter by stating the research questions and hypotheses that guide the rest of the thesis.

Chapter 2 defined the most important terms used in this thesis. We explained which sources of information there are on the Web and reviewed the evaluation measures that we used in subsequent chapters. We finished the background chapter by reviewing related systems that have similar goals.

Chapter 3 introduced the overall architecture of the WebKnox system. We described how components are connected to each other and explained which steps are processed in the main extraction cycle of WebKnox.

Chapter 4 was concerned with finding an update strategy to keep the knowledge base up-to-date without wasting resources. We decided to concentrate on processing news feeds and developed the moving average polling strategy. We also created a large dataset that we used to evaluate our algorithms. We concluded the chapter by stating that the choice of a retrieval algorithm depends on the requirements of the application that uses it. For the purpose of our work, the moving average strategy performed best in terms of low resource consumption and few missed items.

Chapter 5 was concerned with extracting entities from the Web – the main focus of this thesis. We reviewed related work on entity extraction algorithms before introducing five entity extraction techniques that we use in WebKnox. We created techniques that extract entities from HTML structures such as lists and tables, from plain text, and from RDF triples on the Semantic Web. In the last section of this chapter, we compared these extraction techniques against each other, and additionally compared named entity recognition and extraction from the Semantic Web with state-of-the-art algorithms in both fields. We concluded the chapter by saying that despite large differences in precision, all developed extraction techniques are justified.

Chapter 6 was about assessing the extracted entities to increase the precision of the knowledge base. We first identified six different approaches in the related work that could be used for assessing entities before introducing six techniques that we then used for the evaluation.

Chapter 7 described approaches for extracting factual information from the Web. First, we explained how we engineered our ontology that guides the fact extraction process. We then briefly reviewed related work on the subject before we presented five fact extraction algorithms that exploit different source formats on the Web.

Chapter 8 was concerned with the extraction of entity-related information, such as interactive multimedia objects, events, and statements about entities. We reviewed related work, described our own approaches, and evaluated our techniques. Whenever possible, we compared our solutions to state-of-the-art systems.

Chapter 9 was concerned with question answering using the Web and a knowledge base. We reviewed the best-known commercial and educational question answering systems before experimentally developing our own techniques. We then evaluated 13 different approaches to find out which question intent can be best answered using a knowledge base and the Web.

Chapter 10 briefly showed prototypical applications that were created using the techniques and the knowledge base described in previous chapters. We were able to show that the knowledge base is beneficial to both users and machines.

11.2 Fulfilling the Requirements

In Section 1.2, we introduced requirements with which our knowledge base should comply. We will now check which requirements have been met.

Domain Independence

We developed our algorithms to be domain-independent. For the entity and fact extraction evaluation, we used an ontology of 17 concepts ranging from types of people to products to locations. The developed algorithms do not use any domain knowledge unless it was manually assigned for a particular experiment (mentioned in the experimental setup).

Accuracy

We required an extraction accuracy of 80–100 % for both entity and fact extractions. Before entity assessment, the entity extractions are only about 45 % correct on average. The extracted facts are only correct about 60 % of the time. We can increase the precision, however, by assessing entities and using trust thresholds. Using a combined assessor (see Section 6.3.4), we are able to raise the average precision of entities in the knowledge base to over 80 %. We can also increase the fact extraction precision up to about 75 % using a trust threshold. With 75 % precision, we still failed to meet our requirement. Domain-dependent extraction techniques could, however, help improve the extraction precision.

Up-to-Date

We developed an efficient feed polling strategy to retrieve many fresh Web sources in a timely manner. This retrieval strategy is one of the foundations for continuously expanding the knowledge base. We consider this requirement partially met since we did not evaluate whether Web feeds are sufficient to find all new entities. This evaluation was out of the scope of this thesis and is considered a direction for future research.

Semantic

We use an ontology to guide the extraction processes which allows us to export extracted information in RDF triples. This information can be added to the Semantic Web and linked to the Linked Data cloud.

11.3 Answers to Research Questions

We stated our research questions in Section 1.4. Throughout this thesis, we found answers to each of these questions by creating hypotheses, developing prototypes, setting up experiments, and evaluating these hypotheses. This section summarizes the answers to our research questions.

Which techniques can be used to extract entity mentions on the Web and how well do they perform?

Our hypothesis that we can find entities in multiple structures from different sources on the Web is correct. In Chapter 5, we discussed five entity extraction techniques, which we developed to find and extract entities from HTML structures such as lists, from plain text, and from RDF tuples of the Semantic Web. We evaluated how well these techniques perform. While the precision of the techniques ranges from about 15 % (the extraction from plain text using an NER approach) to slightly over 80 % (extracting entities from the Semantic Web), we concluded that all techniques are valuable since there are entities that can often be found with only one of the extraction techniques.

Another conclusion that we draw from the obtained results is that extracted entities need to be assessed in an additional step to filter incorrect extraction results.

Furthermore, our hypothesis stated that using multiple techniques on different sources would yield entity extractions that are not available in DBpedia. One of our experiments showed that almost half of our correct extractions are indeed not available in DBpedia.

How can we efficiently poll Web sources to extract new entities?

We compared several polling strategies and found that a modified moving average polling strategy was the best trade-off between resource limitations and missed items. This line of research was further extended by Reichert (2012), who modified the algorithms presented in this thesis and concluded that they surpass the state-of-the-art polling strategies.

How can we ensure high precision of the extracted information?

We have seen that entity extraction techniques are imprecise (as low as 15 % precision). To filter incorrect extractions, we hypothesized that using only a small set of training data, a set of features, and a supervised machine learning algorithm, we would be able to classify extracted entities as correct or incorrect better than the state-of-the-art algorithms in this field. In Chapter 6, we developed such a setup and showed that it does in fact perform better than all the reviewed techniques, which we implemented and tested on the same data. We developed a combined assessor (using random graph walks and naïve Bayes) to lift the precision of extracted entities to about 80 %.

What entity-centric information is useful for question answering and how can we extract it?

We analyzed hundreds of user questions and found that interactive multimedia objects, factual information, events, statements, and answers can all be related to entities and therefore be of interest to the user. We have shown where each of these information groups can be found and how they can be extracted. Once the information is extracted, it can be connected to a question answering component, which wraps this information into answers.

Our hypothesis stated that we could use a keyword-based approach to find interactive multimedia objects. We applied this approach and were able to show that the techniques yielded 2.55 times more relevant multimedia objects in the top 10 results than our baseline (Google). Concerning the event extraction, we hypothesized that we could use machine learning to extract 5W1H events from news articles. After evaluating this approach we found that about 68 % of the extracted events were at least partially correct. These results are not yet satisfactory but we can say that machine learning is a possible approach for this problem. Regarding the statement extraction, we hypothesized that we would be able to distinguish between positive and negative sentiment of an entity-related statement in 90 % of the cases using text classification. After our evaluation on over 20,000 statements, we reached an accuracy of 89.7 %, which is close to what we expected.

How do ontology-based question answering systems compare to Web extraction-based question answering systems?

Our hypothesis stated that an ontology-based question answering system would answer fewer questions, but would also be more precise than a system that answers questions using the Web. In Chapter 9, we compared 13 different question answering systems to find out whether this thesis was true. The evaluation confirmed our hypothesis; ontology-based systems performed most precisely with 58 % on average, compared to the 49 % precision of Web extraction-based systems. Also as predicted, the recall of these systems (12 %) was lower compared to the recall of extraction-based systems (27 %).

11.4 Contributions

Besides the answers to the posed research questions, we want to reiterate the contributions made in this thesis.

1. **Entity Extraction**: To the best of our knowledge, this is the first thesis on extracting entities from the Web that uses different approaches and compares them against each other. We contribute three new extraction techniques and modify two existing ones (see Chapter 5 and Urbansky et al. (2009a)). Furthermore, we show how one of these extraction algorithms can be used for the task of entity list completion, outperforming the two state-of-the-art approaches Boo!Wa and Google Sets (see Section 5.3.2 and Urbansky et al. (2011d)). Additionally, our named entity recognition technique for extracting entities from plain text uses a novel semi-supervised training approach to learn a model from the Web. Our named entity recognizer can be trained with sparse training data which is impossible for the reviewed state-of-the-art named entity recognizers (see Section 5.2.4 and Urbansky et al. (2011e)). We were also able to show that neither DBpedia nor Freebase contains all the entities we are able to find (see Section 5.3.1).

2. **Entity Assessment**: For entity extraction, there has been no research devoted to comparing different assessment approaches. We compared three successful assessment algorithms reported in the literature and built our own assessment technique upon existing work. Our approach outperforms the best approach from the related work by over 10 % in F1 value (see Chapter 6).

3. **IMO Extraction**: Before this thesis, there was no documented approach on how to extract interactive multimedia objects from the Web. We created an algorithm for extracting this kind of information and outperformed the closest competitor, Google (see Section 8.1 and Werner (2010)).

4. **Fact Extraction**: Despite the plethora of work in information extraction, there are no research results available that compare different fact extraction techniques from the Web. We contribute five techniques for fact extraction and compare them regarding precision and recall (see Chapter 7 and Urbansky et al. (2008)).

5. **Feed Polling Strategy**: We contributed a polling strategy for Web feeds, which is superior to other state-of-the-art strategies under certain criteria (see Chapter 4 and Urbansky et al. (2011c), Reichert et al. (2011) and Reichert (2012)).

6. **Question Answering**: To the best of our knowledge, we created the most comprehensive evaluation of 13 question answering approaches on a set of real world questions. We also contributed three approaches to question answering in this thesis (see Chapter 9 and Urbansky et al. (2009b)).

7. **Ontology Creation**: WebKnox both uses and enables Ontofly, a Web-based ontology creation tool (see Urbansky et al. (2010), Willner (2010), and Willner (2011)).

8. **Datasets**: For all evaluations in this thesis, we tried to fairly compare different algorithms on the same datasets. Whenever possible and reasonable, we used existing datasets. Often, however, researchers do not make their test data available, making a fair comparison difficult. For several evaluations we therefore created our own datasets to allow for a fair evaluation. We made these datasets publicly available on the dataset platform Areca (Urbansky et al., 2011b). These datasets are summarized in the following list.

   (a) Web Feeds Items[1]
   (b) Sentiment Statements[2]
   (c) Web Named Entity Recognition[3]
   (d) Question Answering[4]

9. **Palladian**: Many of the algorithms introduced in this thesis are now freely available for research purposes in the Java library Palladian (Urbansky et al., 2011a). This allows other researchers to compare their algorithms in a fair manner to our approaches and advance research in this field.

11.5 Future Work

In the process of answering the posed research questions, many new questions have arisen that could not all be answered within the scope of this thesis. We now outline directions for future research.

11.5.1 Knowledge Base Expansion

The number of entities is growing on a daily basis, so we have to find ways of extracting new entities quickly. In this thesis, we have shown an efficient news feed polling strategy which makes it possible to read and scan thousands of news articles for entity mentions. We have to

[1]`http://areca.co/1/Feed-Item-Dataset-TUDCS1`, last accessed on 20th of March 2012
[2]`http://areca.co/16/Short-Opinionated-Sentences-about-Diverse-Topics`, last accessed on 20th of March 2012
[3]`http://areca.co/7/Web-Named-Entity-Recognition-TUDCS4`, last accessed on 20th of March 2012
[4]`http://areca.co/17/600-random-questions-from-six-datasets`, last accessed on 20th of March 2012

evaluate, however, how many new entities are indeed mentioned in news feeds and whether other sources are necessary to quickly find new entities. We have not studied entity extraction from the Deep Web in this thesis, although the size of the Deep Web makes it a potentially good source for extracting entities. The approaches for extracting entities from the Deep Web will likely differ from the domain-independent techniques presented in this work.

11.5.2 Domain-dependent Extraction Algorithms

We have researched domain-independent extraction algorithms in this thesis. The domain independence comes at the cost of a reduced extraction precision. Knowing about the domain or, more specifically, about the concept for which entities are extracted, could improve extraction precision considerably. For example, knowing the textual contexts in which entities of a particular concept appear enables phrase extraction algorithms with a higher recall than generic phrases, such as "movies like `X`". Also, we could improve fact extraction accuracy by allowing extraction only from trustworthy sources or even limiting extraction to pages that follow a certain pattern. We have used this approach for extraction from QA-rich websites and learned that this method can yield high extraction precision at the cost of manual labor beforehand.

11.5.3 Linked Data Cloud Connections

Our knowledge base can be represented in RDF, which makes it a contribution to the Semantic Web movement. A successful integration of our knowledge base into the Linked Data cloud requires much more effort. Most importantly, we need to match schemata and connect entities to identical resources in other datasets. A first step would be to use heuristics to automatically link popular entities to well-known Linked Data repositories, such as DBpedia, MusicBrainz, and Freebase.

The Web is a vibrant network of people and information. Structuring, filtering, and mining information from the immense stream of new data that becomes available on the Web every second is one of the most interesting and challenging tasks in the next few years.

Appendix A

Manually-assigned RDF Classes for Semantic Web Extraction Evaluation

Concept	Semantic Web URI
Actor	http://dbpedia.org/ontology/Actor
	http://www.mpii.de/yago/resource/wordnet_actor_10976527
	http://data.linkedmdb.org/resource/movie/actor
	http://rdf.freebase.com/ns/film.actor
	http://rdf.freebase.com/ns/tv.tv_actor
	http://sw.opencyc.org/concept/Mx4rvVjaHZwpEbGdrcN5Y29ycA
Airplane	http://rdf.freebase.com/ns/aviation.aircraft_model
	http://sw.opencyc.org/concept/Mx4rvViuUJwpEbGdrcN5Y29ycA
	http://dbpedia.org/resource/Category:Low_wing_aircraft
	http://dbpedia.org/resource/Category:Multiple_engine_aircraft
	http://dbpedia.org/resource/Category:Jet_aircraft
Airport	http://airports.dataincubator.org/schema/LargeAirport
	http://airports.dataincubator.org/schema/SmallAirport
	http://dbpedia.org/ontology/Airport
	http://sw.opencyc.org/concept/Mx4rvVj-r5wpEbGdrcN5Y29ycA
Athlete	http://dbpedia.org/ontology/Athlete
	http://sw.opencyc.org/concept/Mx4rvVi--5wpEbGdrcN5Y29ycA
	http://rdf.freebase.com/ns/sports.pro_athlete
Band	http://musicbrainz.org/mm/mm-2.1#Artist
	http://purl.org/ontology/mo/MusicArtist
	http://purl.org/ontology/mo/MusicGroup

	http://dbpedia.org/ontology/Band
	http://sw.opencyc.org/concept/Mx4rvgEZGJwpEbGdrcN5Y29ycA
Car	http://rdf.freebase.com/ns/base.sportscars.sports_car
	http://sw.opencyc.org/concept/Mx4rvViVwZwpEbGdrcN5Y29ycA
	http://dbpedia.org/ontology/Automobile
City	http://www.mpii.de/yago/resource/wordnet_city_108524735
	http://rdf.freebase.com/ns/location.citytown
	http://semanticweb.org/id/Category-3ACity
	http://dbpedia.org/ontology/City
Comp. Mouse	http://sw.opencyc.org/concept/Mx4rvVjLo5wpEbGdrcN5Y29ycA
	http://dbpedia.org/resource/Category:Computing_input_devices
Country	http://rdf.freebase.com/ns/location.country
	http://www4.wiwiss.fu-berlin.de/factbook/ns#Country
	http://semanticweb.org/id/Category-3ACountry
	http://sw.opencyc.org/concept/Mx4rvViIeZwpEbGdrcN5Y29ycA
	http://airports.dataincubator.org/schema/country
	http://dbpedia.org/ontology/Country
Lake	http://rdf.freebase.com/ns/geography.lake
	http://dbpedia.org/ontology/Lake
	http://sw.opencyc.org/concept/Mx4rvVi4IpwpEbGdrcN5Y29ycA
Mobile Phone	http://sw.opencyc.org/concept/Mx4rvVjj6pwpEbGdrcN5Y29ycA
	http://dbpedia.org/resource/Category:Smartphones
Movie	http://rdf.freebase.com/ns/film.film
	http://data.linkedmdb.org/resource/movie/film
	http://dbpedia.org/ontology/Film
	http://sw.opencyc.org/concept/Mx4rv973YpwpEbGdrcN5Y29ycA
Newspaper	http://sw.opencyc.org/concept/Mx4rv4ByW5wpEbGdrcN5Y29ycA
	http://rdf.freebase.com/ns/book.periodical
	http://rdf.freebase.com/ns/book.newspaper
	http://dbpedia.org/ontology/Newspaper
Politician	http://dbpedia.org/ontology/Politician
	http://sw.opencyc.org/concept/Mx4rvVjntpwpEbGdrcN5Y29ycA
	http://rdf.freebase.com/ns/government.politician
Restaurant	http://rdf.freebase.com/ns/base.popstra.restaurant
	http://sw.opencyc.org/concept/Mx4rLYFgxAhuEduVMwDggVaqog
	http://dbpedia.org/resource/Category:Multinational_food_companies

	http://dbpedia.org/resource/Category:Fast-food_franchises
	http://dbpedia.org/resource/Category:Fast-food_hamburger_restaurants
Sports Team	http://dbpedia.org/ontology/SportsTeam
	http://sw.opencyc.org/concept/Mx4rvViwbJwpEbGdrcN5Y29ycA
	http://rdf.freebase.com/ns/sports.sports_team
University	http://dbpedia.org/ontology/University
	http://rdf.freebase.com/ns/education.university
	http://sw.opencyc.org/concept/Mx4rvVjjvpwpEbGdrcN5Y29ycA

Table A.1: Manually-assigned URIs from the Semantic Web to the 17 Concepts Used for the Evaluation

Appendix B

Trust Threshold Analysis of Combined Assessors

In Section 6.2.6, we argue that the combined assessor using RGW+NB is superior to other combinations. Looking at Table 6.2 does not provide evidence to our claim as the combination RGW+NB+Text yields the highest precision and the combination NB+KNN yields the highest F1 value and accuracy. The values from the table are, however, only one possible setting of the trust threshold that we can use with the combined assessors. The following figures show the trust threshold analysis for all five combinations. We decided to use RGW+NB because it is the only combination for which the precision can increase with a trust threshold > 0.5 without an immediate decrease in accuracy.

The NB+KNN combination (see Figure B.1 – the combinations NB+Text and NB+KNN+Text behave very similarly, see Figure B.2 and B.3 respectively) has the potential to reach a precision of about 90 % at a trust threshold of 0.9, but at the cost of a recall of only slightly over 35 %. The RGW+NB combination (see Figure B.5) reaches roughly the same level of precision at a lower trust threshold of 0.7 and therefore yields a higher recall at that level.

The RGW+NB+Text combination (see Figure B.4) reaches the 90 % precision level even earlier at a trust threshold of about 0.6. Due to the combination of three classifiers that all have to independently classify the entity as correct for it to be finally classified as correct, the drop in recall is sharp with increasing trust thresholds.

In conclusion, we can say that combining RGW+NB results in a combination that reaches a precision level of 90 % with the highest accuracy and is therefore considered best.

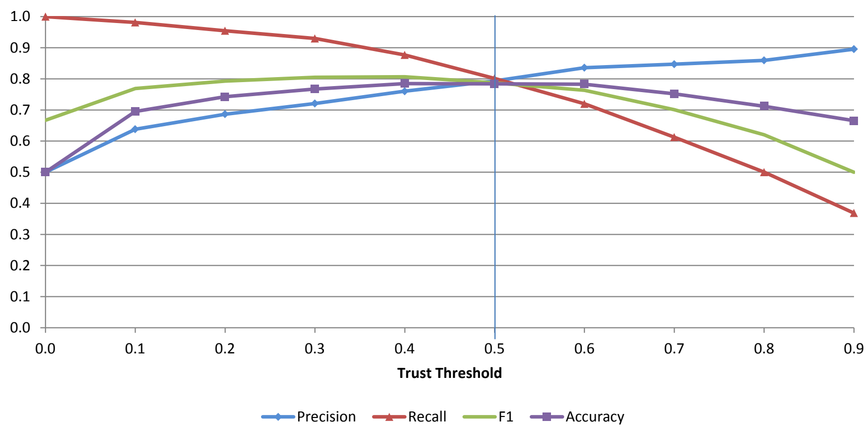


Figure B.1: Threshold Analysis of the Combined Assessor (NB + KNN)

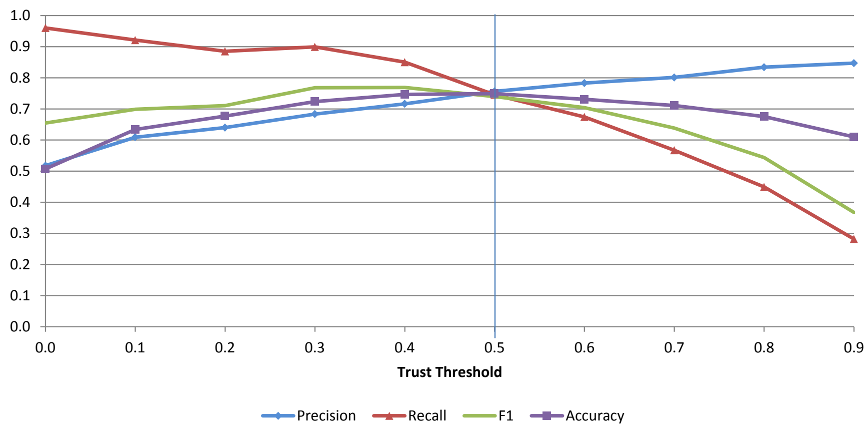


Figure B.2: Threshold Analysis of the Combined Assessor (NB + Text)

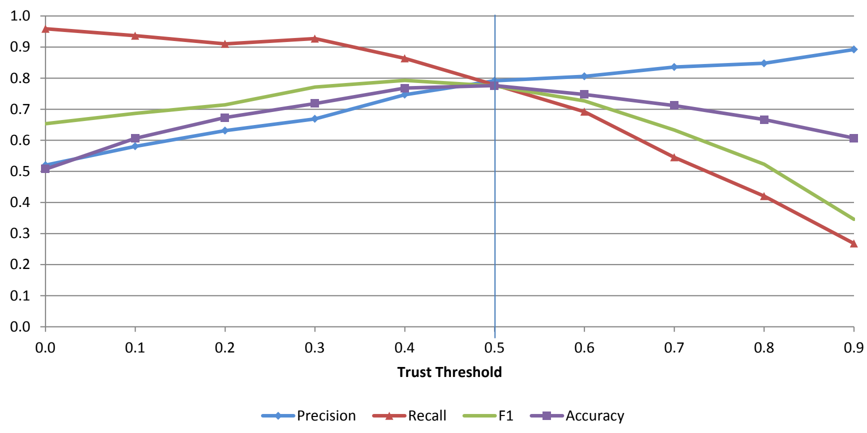


Figure B.3: Threshold Analysis of the Combined Assessor (NB + KNN + Text)

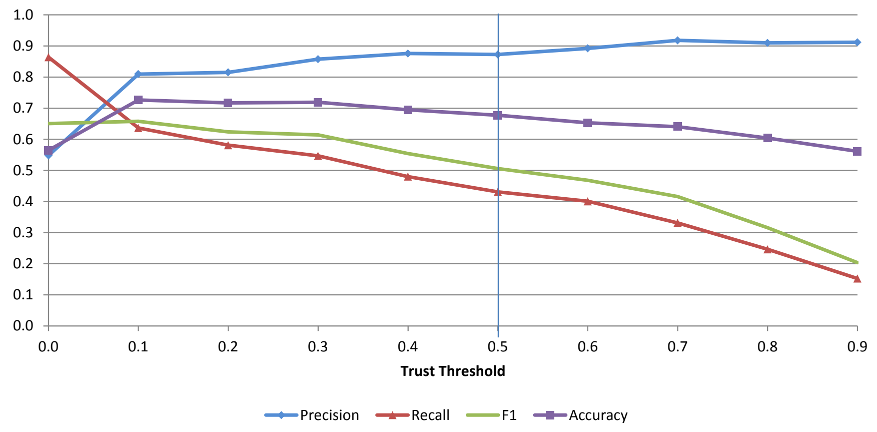


Figure B.4: Threshold Analysis of the Combined Assessor (RGW + NB + Text)

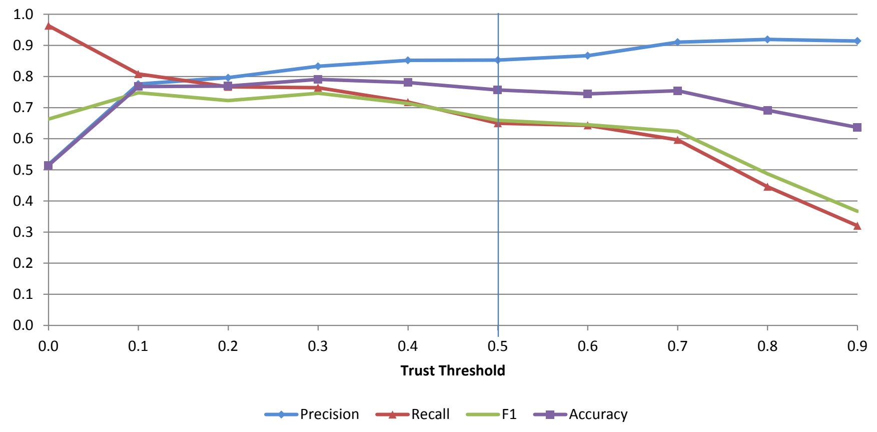


Figure B.5: Threshold Analysis of the Combined Assessor (RGW + NB)

Appendix C

Evaluation Entities for Fact Extraction

Concept	**Entities**
Actor	Jim Carrey, Mel Gibson, Laura Dern, Monica Potter, James Stewart, Ken Watanabe, Orlando Bloom, Tom Wilkinson, Chace Crawford, Natalie Portman
Airplane	IAR 95, Reggiane Re 2001, Folland Gnat, Dassault Mirage IIIV, Bristol Scout, North American FJ-1 Fury, Douglas F5D Skylancer, Parnall Plover, Sukhoi Su-7, Boeing P-12
Airport	Hong Kong International Airport, Heathrow Airport, Detroit Metropolitan Wayne County Airport, George Bush Intercontinental Airport, Frankfurt Airport, John F. Kennedy International Airport, O'Hare International Airport, Paris-Charles de Gaulle Airport, Dubai International Airport, Orlando International Airport
Athlete	Kareem Abdul Jabbar, Peyton Manning, Davor Suker, Mark Messier, Babe Ruth, Willie Mays, Michael Jordan, Aaron Rodgers, Michael Ballack, Wayne Getzky
Band	Led Zeppelin, The Beatles, Pink Floyd, The Jimi Hendrix Experience, Van Halen, Queen, Eagles, Metallica, U2, Bob Marley and the Wailers
Car	2006 Bugatty Veyron16.4, 2008 Lamborghini Reventon, 2009 Jaguar XF, 2012 Lotus Exige, 2009 Tesla Roadster, 2009 Maserati Grand Turismo S, 2009 Aston Martin VS Vantage, 2012 Fiat 500, 2012 Audi R8 Spyder, 2011 Bentley Continental GTC
City	Gustavia, Brasilia, Jerusalem, Tallin, Baku, Tashkent, Damascus, Cairo, Tripoli, Beirut

Comp. Mouse	Microsoft Wireless Mouse 5000, Microsoft Arc Mouse, Microsoft Natural Wireless Laser Mouse 6000, Microsoft Wireless Mobile Mouse 6000, Verbatim Wireless Optical Touch Mouse 97564, Logitech M305 Wireless Mouse, Nexus SM-7000B Silent Mouse, Kensington K72356US, 2.4G Nano Wireless Blue Optical/light Mouse, Saitek W07 Touch Force Gaming Mouse
Country	China, Italy, Nepal, Malawi, Austria, Finland, Kuwait, Cyprus, Vanuatu, Tuvalu
Lake	Lake Lugano, Lake Champlain, Issyk Kul, Kaptai Lake, Lake Balaton, Lake Chad, Lake Kaindy, Sea of Galilee, Lake Kivu, Lake Assad
Mobile Phone	BlackBerry Bold Touch 9000, Apple iPhone 4S, Alcatel OT-807, Dell Venue, Sony Ericsson Cedar, Icemobile Tornado II, Samsung Galaxy SII Epic 4G Touch, LG Optimus 2X SU 660, Philips W625, Nokia Lumia 800
Movie	Braveheart, The Dark Knight, Idiocracy, Code 46, Iron Man, The Descendants, A Dangerous Method, Super 8, Tower Heist, Puss in Boots
Newspaper	The Wall Street Journal, The New York Times, The Washington Post, New York Post, Chicago Tribune, The Philadelphia Inquirer, The Denver Post, Star Tribune, The Plain Dealer, Chicago Sun-Times
Politician	Carlos P. Romulo, Eelco van Kleffens, Barack Obama, Cheddi Jagan, Angela Merkel, Osvaldo Aranha, Dmitry Medvedev, Anders Fogh Rasmussen, Giorgio Napolitano, Donna Shalala
Restaurant	Pollo Campero, MOS Burger, Shakey's Pizza, Hard Rock Cafe, Red Lobster, The Keg, Carl's Jr., Quiznos, Telepizza, Panera Bread
Sports Team	Manchester United, Dallas Cowboys, New York Yankees, Washington Redskins, Real Madrid, New England Patriots, New York Giants, Houston Texans, New York Jets, Arsenal F.C.
University	Harvard University, California Institute of Technology, University of Toronto, University of California, Berkeley, Tsinghua University, University of Michigan, University of Washington, University of Alberta, University of York, Emory University

Table C.1: Entities for the Fact Extraction Evaluation

Appendix D

Datatype Mapping for Fact Extraction

Datatype	**Regular Expression**
Numeric	`(?<!(\w )-)(?<!(\w ))-?((\\d){1,}((,|\.|\s ))?){1,} (?!((\d )+-(\\d)+))(?!-(\d )+)`
Boolean	`(?<!(\w ))(?i)(yes|no|true|false|n.a.) (?!(\w ))`
String	`([A-Z.]{1}([A-Za-z-0-9.]*)(\s )?)+   ([A-Z.0-9]+([A-Za-z-0-9.]*) (\s )?)*`
AnyURI	`((http://|www.).*?(?=[.,;?!]?           (\s |\]|\))|[.,;?!]?$))| ([A-Za-z.0-9-]*?            \\.(de|com|cc|tv|us|net|org|gov|mil |edu|fr|it|com.au|co.uk)[/A-Za-z0-9-]* (\.[A-Za-z]{2,5})?)`
Date	`((\d ){4}-(\d ){2}-(\d ){2})|((\d ){1,2} [\.|/|-](\\d){1,2}[\.|/|-](\d ){1,4})           |((?<!(\d ){2}) (\d ){1,2}(th)?(\.)?(\s )?    ([A-Za-z]){3,9}((\,)|(\s ))+([’])? (\d ){2,4})|((\w ){3,9}\s (\d ){1,2}(th)? ((\,)|(\s ))+([’])?(\d ){2,4})`
AnyType	`(.)*`

Table D.1: Regular Expressions for Attribute Datatypes

Appendix E

Statement Extraction Entities and Results

Concept	**Entities**
Product	Palm Pre, MacBook Pro, Bugatti Veyron 16.4, Volvo C30 BEV, The Lost Symbol
Location	San Francisco, Frankfurt, Riesa, Gilroy, Luxor
Person	Barack Obama, Amy MacDonald, Robin Williams, Bill Gates, Reiner Kraft
Organization	Yahoo Inc., AT&T Inc., Rotary International, IKEA, Live Like A German

Table E.1: Entities for the Statement Extraction Evaluation

Measure	**Assignment**	**Google Web**	**Editorial Pick**	**WebKnox**
Relevance	Relevant	29.6 %	**68.3 %**	56.2 %
	Somewhat Relevant	34.2 %	27.1 %	25.2 %
	Not Relevant	36.2 %	4.6 %	18.7 %
Interestingness	Interesting	18.5 %	**55.8 %**	41.4 %
	Somewhat Interesting	36.0 %	68.3 %	32.7 %
	Not Interesting	52.6 %	8.2 %	25.8 %
Curiosity	Learn More	27.8 %	**67.4 %**	40.7 %
	Not Learn More	72.2 %	32.6 %	59.3 %

Table E.2: Evaluation Results of the Statement Extraction

Bibliography

George Adam, Christos Bouras, and Vassilis Poulopoulos. Efficient extraction of news articles based on RSS crawling. In *Proceedings of the International Conference on Machine and Web Intelligence*, 2010.

Adobe. http://www.adobe.com/devnet/flashplayer/articles/swf_searchability.html, 2008.

Alexander Afanasyev, Jiangzhe Wang, Chunyi Peng, and Lixia Zhang. Measuring redundancy level on the Web. In *Proceedings of the 7th Asian Internet Engineering Conference*, 2011.

Eugene Agichtein, Luis Gravano, Jeff Pavel, Viktoriya Sokolova, and Aleksandr Voskoboynik. Snowball: A prototype system for extracting relations from large text collections. *ACM SIGMOD Record*, 2001.

Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning to find answers to questions on the web. *ACM Transactions on Internet Technology*, 2004.

Sanjay Agrawal, Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti, Arnd Christian König, and Dong Xin. Exploiting web search engines to search structured databases. In *Proceedings of the 18th International Conference on World Wide Web*, 2009.

Enrique Alfonseca and Maria Ruiz-casado. Learning sure-fire rules for named entities recognition. In *Proceedings of the International Workshop in Text Mining Research, Practice and Opportunities, in conjunction with RANLP Conference*, 2005.

Alias-i. LingPipe 4.0.1, 2011. URL {`http://alias-i.com/lingpipe`}.

James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic Detection and Tracking Pilot Study. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

Allied Business Intelligence. Mobile Data Usage Grows Exponentially but Data Revenue Lags, 2010.

Bengt Altenberg. Causal linking in spoken and written English. *Studia Linguistica*, 1984.

Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of Human Language Technology Conference*, 2003.

Martin Atkinson, Jakub Piskorski, Bruno Pouliquen, Ralf Steinberger, Hristo Tanev, and Vanni Zavarella. Online-monitoring of security-related events. In *Proceedings of the 22nd International Conference on Computational Linguistics: Demonstration Papers*, 2008.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference*, 2007.

Dominic Balasuriya, Nicky Ringland, Joel Nothman, Tara Murphy, and James R. Curran. Named entity recognition in Wikipedia. In *Proceedings of the Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, 2009.

Krisztian Balog, Edgar Meij, and Maarten de Rijke. Entity Search: Building Bridges between Two Worlds. In *Proceedings of the Semantic Search Workshop at WWW*, 2010a.

Krisztian Balog, Pavel Serdyukov, and Arjen P. de Vries. Overview of the TREC 2010 Entity Track. In *TREC 2010 Working Notes*, 2010b.

Michele Banko and Oren Etzioni. Strategies for Lifelong Knowledge Extraction from the Web. In *Proceedings of the 4th International Conference on Knowledge Capture*, 2007.

Michele Banko, Micheal J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-Web entry points. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.

Michael K. Bergman. The Deep Web: Surfacing Hidden Value. *The Journal of Electronic Publishing*, 2001.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001.

Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, 1997.

Chris Bizer. DBpedia 3.7 released, including 15 localized Editions, 2011. URL `{http://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/}`.

Duncan Bloor. The most asked questions on Google, 2011. URL `{http://searchinsights.wordpress.com/2011/09/01/the-most-asked-questions/}`.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Conference on Computational Learning Theory*, 1998.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A Collaboratively Created Graph Database For Structuring Human Knowledge. In *Proceedings of the 2008 SIGMOD International Conference on Management of Data*, 2008.

Oriol Borrega, Mariona Taulé, and M. Antø'nia Martı. What do we mean when we speak about Named Entities. In *Proceedings of Corpus Linguistics*, 2007.

Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. NYU: Description of the MENE named entity system as used in MUC-7. In *Proceedings of the 7th Message Understanding Conference*, 1998.

Laura Bright, Avigdor Gal, and Louiqa Raschid. Adaptive pull-based policies for wide area data delivery. *ACM Transactions Database Systems*, 2006.

Sergey Brin. Extracting Patterns and Relations from the World Wide Web. *WebDB Workshop at 6th International Conference on Extending Database Technology*, 1998.

Andrei Broder. A taxonomy of web search. In *SIGIR Forum*, 2002.

Jeen Broekstra and Arjohn Kampman. Sesame: A generic Architecture for Storing and Querying RDF and RDF schema. *The Semantic WebISWC 2002*, 2002.

Marc Bron, Krisztian Balog, and Maarten de Rijke. Related Entity Finding Based on Co-Occurrence. In *Proceedings of the 18th Text REtrieval Conference*, 2010.

Ada Brunstein. Annotation guidelines for answer types. *Linguistic Data Consortium*, 2002. URL `http://www.ldc.upenn.edu/Catalog/docs/LDC2005T33/BBN-Types-Subtypes.html`.

Sabine Buchholz and Antal van den Bosch. Integrating seed names and n-grams for a named entity list and classifier. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, 2000.

Robin D. Burke, Kristian J. Hammond, Vladimir Kulyukin, Steven L. Lytinen, Noriko Tomuro, and Scott Schoenberg. Question Answering from Frequently Asked Question Files: Experiences with the FAQFinder system. *AI Magazine*, 1997.

Jamie Callan and Teruko Mitamura. Knowledge-based extraction of named entities. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, 2002.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the 24th Conference on Artificial Intelligence*, 2010a.

Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka Jr., and Tom M. Mitchell. Coupled Semi-Supervised Learning for Information Extraction. In *Proceedings of the 3rd International Conference on Web Search and Data Mining*, 2010b.

Francesca Carmagnola. The five Ws in user model interoperability. In *Workshop on Ubiquitous User Modeling at the Intelligent User Interfaces Conference*, 2008.

Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate track papers & posters*, 2004.

Chia-Hui Chang, Mohammed Kayed, Mohed R. Girgis, and Khaled F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. Exploiting web search to generate synonyms for entities. In *Proceedings of the 18th International Conference on World Wide Web*, 2009.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Artificial Intelligence Research*, 2002.

Nancy Chinchor. MUC-7 Named Entity Task Definition. In *Proceedings of the 7th Message Understanding Conference*, 1997. URL {`http://acl.ldc.upenn.edu/muc7/ne\_task.html`}.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

David Chmielewski and Gongzhu Hu. A Distributed Platform for Archiving and Retrieving RSS Feeds. In *Proceedings of the 4th ACIS International Conference on Computer and Information Science*, 2005.

Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for Web crawlers. *ACM Transactions Database Systems*, 2003.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 1990.

James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. Technical report, World Wide Web Consortium, 1999. URL {`http://www.w3.org/TR/xpath`}.

William W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

William W. Cohen and Wei Fan. Learning page-independent heuristics for extracting data from web pages. *Computer Networks-the International Journal of Computer and Telecommunications Networking*, 1999.

William W. Cohen, Matthew Hurst, and Lee S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of the 11th International Conference on World Wide Web*, 2002.

Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting of Association for Computational Linguistics, July*, 2002.

Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yueheng Sun. Finding question-answer pairs from online forums. In *Proceedings of the 31st annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008.

Dan Crow. Google Squared: Web scale, open domain information extraction and presentation. In *Proceedings of the 32nd European Conference on Information Retrieval, Industry Day*, 2010.

Alessandro Cucchiarelli and Paola Velardi. Unsupervised Named Entity Recognition Using Syntactic and Semantic Contextual Evidence. *Computational Linguistics*, 2001.

Silviu Cucerzan and David Yarowsky. Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence. In *Proceedings of the Workshop on Very Large Corpora at the Conference on Empirical Methods in NLP*, 1999.

Richard Cyganiak and Anja Jentzsch. Linking Open Data cloud diagram, 2011. URL `{http://lod-cloud.net/}`.

Bhavana Dalvi, Jamie Callan, and William Cohen. Entity List Completion Using Set Expansion Techniques. In *Proceedings of the 19th Text REtrieval Conference*, 2011.

Pedro DeRose, Warren Shen, Fei Chen, Yoonkyong Lee, Doug Burdick, AnHai Doan, and Raghu Ramakrishnan. DBLife: A Community Information Management Platform for the Database Research Community (Demo). In *CIDR*, 2007.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.

Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proceedings of the 1st International Conference on Autonomous Agents*, 1997.

Doug Downey. *Redundancy in Web-scale Information Extraction: Probabilistic Model and Experimental Results*. PhD thesis, University of Washington, 2008.

Doug Downey, Oren Etzioni, Stephen Soderland, and Daniel S. Weld. Learning Text Patterns for Web Information Extraction and Assessment. In *Workshop on Adaptive Text Extraction and Mining*, 2004.

Doug Downey, Oren Etzioni, and Stephen Soderland. A Probabilistic Model of Redundancy in Information Extraction. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

Doug Downey, Matthew Broadhead, and Oren Etzioni. Locating complex named entities in web text. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2007.

Doug Downey, Oren Etzioni, and Stephen Soderland. Analysis of a Probabilistic Model of Redundancy in Unsupervised Information Extraction. *Artificial Intelligence*, 2010.

Jenny Edwards, Kevin McCurley, and John Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th International Conference on World Wide Web*, 2001.

Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th International Conference on World Wide Web*, 2004.

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Steven Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, 2005.

Richard Evans. A framework for named entity recognition in the open domain. *Recent advances in natural language processing III: selected papers from RANLP 2003*, 2003.

Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

Elena Filatova and Vasileios Hatzivassiloglou. Event-Based Extractive Summarization. In *Proceedings of the ACL Workshop*, 2004.

Sanda Harabagiu Finley and Sanda M. Harabagiu. Generating single and multi-document summaries with gistexter. In *Proceedings of the Workshop on Automatic Summarization*, 2002.

Michael Fleischman and Eduard Hovy. Fine Grained Classification of Named Entities. In *Proceedings of the 19th International Conference on Computational Linguistics*, 2002.

Joseph L. Fleiss. The Measurement of Interrater Agreement. *Statistical Methods for Rates and Proportions*, 1981.

Radu Florian. Named entity recognition as a house of cards: Classifier stacking. In *Proceedings of the 6th conference on Natural language learning*, 2002.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named Entity Recognition through Classifier Combination. In *Proceedings of CoNLL*, 2003.

Charles Frankel, Michael J. Swain, and Vassilis Athitsos. WebSeer: An Image Search Engine for the World Wide Web. Technical report, University of Chicago, 1996.

Eric Franzon. LOD Cloud Updated - Time to Change Your Slide Decks! Blog Article, 2011. URL {`http://semanticweb.com/lod-cloud-updated-time-to-change-your-slide-decks_b23252`}.

Christopher Friedrich. WebSnippets: Extracting and Ranking of entity-centric knowledge from the Web. Master's thesis, Dresden University of Technology, 2010.

Marcel Gerlach. Verbesserung von Question/Answering mithilfe von Suchmaschinenanfragen. Master's thesis, Dresden University of Technology, 2012.

Google. Google Sets. System and methods for automatically creating lists, March 2008.

Google. pubsubhubbub, 2010. URL {`http://code.google.com/p/pubsubhubbub/`}.

Carrie Grimes. Microscale evolution of web pages. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.

Ralph Grishman and Beth Sundheim. Message Understanding Conference-6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics*, 1996.

Ralph Grishman, Silja Huttunen, and Roman Yangarber. Real-time event extraction for infectious disease outbreaks. In *Proceedings of the 2nd International Conference on Human Language Technology Research*, 2002.

Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd International SIGIR Conference on Research and Development in Information Retrieval*, 2009.

Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004.

Young G. Han, Sang H. Lee, Jae H. Kim, and Yanggon Kim. A new aggregation policy for RSS services. In *Proceedings of the International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation*, 2008.

Martin A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics*, 1992.

Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Proceedings of the 4th International Conference on Semantic and Digital Media Technologies*, 2009.

Christian Hensel. Automatische Fließtexterstellung aus Entitätsfakten in einer Wissensbasis. Master's thesis, Dresden University of Technology, 2011.

Chih Hu and Chung-Kuang Chou. RSS watchdog: an instant event monitor on real online news streams. In *Proceedings of the 18th Conference on Information and Knowledge Management*, 2009.

Fei Huang. *Multilingual named entity extraction and translation from text and speech.* PhD thesis, Carnegie Mellon University, 2005.

Matthew Hurst and Alexey Maykov. Social Streams Blog Crawler. In *Proceedings of the 25th International Conference on Data Engineering*, 2009.

Francisco Iacobelli, Nathan Nichols, and Larry Birnbaum Kristian Hammond. Finding new information via robust entity detection. In *Proceedings of Proactive Assistant Agents*, 2010.

Alpa Jain and Marco Pennacchiotti. Open entity extraction from web search query logs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010.

Bernard J. Jansen, Danielle Booth, and Amanda Spink. Determining the informational, navigational, and transactional intent of Web queries. *Information Processing & Management*, 2008.

Heng Ji and Ralph Grishman. Refining Event Extraction through Cross-Document Inference. In *Proceedings of ACL*, 2008.

Valentin Jijkoun and Maarten de Rijke. Retrieving answers from frequently asked questions pages on the web. In *Proceedings of the International Conference on Information and Knowledge Management*, 2005.

Vangelis Karkaletsis, Georgios Paliouras, Georgios Petasis, Natasa Manousopoulou, and Constantine D. Spyropoulos. Named-entity recognition from Greek and English texts. *Journal of Intelligent and Robotic Systems*, 1999.

Gjergji Kasneci, Maya Ramanath, Fabian M. Suchanek, and Gerhard Weikum. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Record*, 2008.

Boris Katz. Annotating the World Wide Web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet*, 1997.

Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy J. Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. *Lecture notes in computer science*, 2002.

Jun'ichi Kazama and Kentaro Torisawa. Exploiting Wikipedia as external knowledge for named entity recognition. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

Andy King. The Average Web Page. Blog Entry, 2009. URL `{http://www.optimizationweek.com/reviews/average-web-page/}`.

Gary King and Will Lowe. An automated information extraction tool for international conflict data with performance as good as human coders: A rare events evaluation design, International Organization. *International Organization*, 2003.

Dan Klein, Joseph Smarr, Huy Nguyen, and Christopher D. Manning. Named entity recognition with character-level models. In *Proceedings of CoNLL*, 2003.

Zornitsa Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, 2006.

Zornitsa Kozareva, Boyan Bonev, and Andres Montoyo. Self-training and co-training applied to spanish named entity recognition. *Advances in Artificial Intelligence*, 2005.

Saul A. Kripke. *Naming and necessity*. Wiley-Blackwell, 1981.

Cody C. Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the Web. In *Proceedings of the 10th International Conference on World Wide Web*, 2001.

J. Richard Landis and Gary G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 1977.

Joseph LaPorte. Rigid Designators. Stanford Encyclopedia of Philosophy, 2006. URL `{http://plato.stanford.edu/entries/rigid-designators/}`.

Paul J. Leach, Michael Mealling, and Rich Salz. RFC4122 - A Universally Unique IDentifier (UUID) URN Namespace. Technical report, Microsoft Inc., Refactored Networks, LLC, DataPower Technology, Inc., 2005. http://www.faqs.org/rfcs/rfc4122.html.

Bum-Suk Lee, Jin W. Im, Byung-Yeon Hwang, and Du Zhang. Design of an RSS Crawler with Adaptive Revisit Manager. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering*, 2008.

Bumsuk Lee and Byung-Yeon Hwang. An Efficient Method Predicting Update Probability on Blogs. In *Proceedings of the International Conference on Mathematics and Computers in Science and Engineering*, 2009.

Jongwuk Lee, Seung won Hwang, Zaiqing Nie, and Ji-Rong Wen. Navigation System for Product Search. In *Proceedings of the 26th International Conference on Data Engineering*, 2010.

Jens Lehmann, Jörg Schüppel, and Sören Auer. Discovering Unknown Connections - the DBpedia Relationship Finder. In *Proceedings of the 1st Conference on Social Semantic Web*, 2007.

Gennadiy Lemberski. Named entity recognition in Hebrew language; Hebrew Multiword Expression: approaches and recognition methods. Master's thesis, Ben-Gurion University of the Negev, 2003. URL `http://www.cs.bgu.ac.il/~gennadaa/thesis.pdf`.

Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 1995.

Wenjie Li, Mingli Wu, Qin Lu, Wei Xu, and Chunfa Yuan. Extractive summarization using inter- and intra-event relevance. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics*, 2006.

Jimmy Lin, Aaron Fernandes, Boris Katz, Gregory Marton, and Stefanie Tellex. Extracting answers from the web using knowledge annotation and knowledge mining techniques. Technical report, DTIC Document, 2006.

Yi-Chung Lin and Peng-Hsiang Hung. Probabilistic named entity verification. In *Proceedings of the 2nd International Workshop on Computational Terminology*, 2002.

Roger Lindsay and Barbara Gorayska. Relevance, Goals and Cognitive Technology. *International Journal of Cognitive Technology*, 2002.

LingPipe. Genes are Generic, People are Specific. Blog Post, 2007. http://lingpipe-blog.com/2007/08/23/genes-are-generic-people-are-specific/.

Hongzhou Liu, Venugopalan Ramasubramanian, and Emin Gün Sirer. Client behavior and feed characteristics of RSS, a publish- subscribe system for web micronews. In *Proceedings of the 5th SIGCOMM Conference on Internet Measurement*, 2005.

Jiahui Liu and Larry Birnbaum. What do they think?: aggregating local views about news events and topics. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.

Maofu Liu, Wenjie Li, Mingli Wu, and Qin Lu. Extractive Summarization Based on Event Term Clustering. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, 2007.

Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing Named Entities in Tweets. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

Vanessa Lopez, Enrico Motta, and Victoria Uren. Poweraqua: Fishing the semantic web. *The Semantic Web: Research and Applications*, 2006.

Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. AquaLog: An ontology-driven question answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2007.

Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's Deep-Web Crawl. *Proceedings of the VLDB Endowment*, 2008.

Christopher D. Manning. Doing named entity recognition? don't optimize for f1. Website, Blog, 2006. URL `http://nlpers.blogspot.com/2006/08/doing-named-entity-recognition-dont.html`.

Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing.* MIT Press, 2002.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

Mónica Marrero, Sonia Sánchez-Cuadrado, Jorge Morato Lara, and George Andreadakis. Evaluation of named entity extraction systems. *Advances in Computational Linguistics, Research in Computing Science*, 2009.

Elaine Marsh and Dennis Perzanowski. MUC-7 Evaluation of IE Technology: Overview of Results. In *Proceedings of the 7th Message Understanding Conference*, 1998.

Johanna May. Definition: knowledge base, 2001. URL `{http://searchcrm.techtarget.com/definition/knowledge-base}`.

Diana Maynard, Valentin Tablan, Cristian Ursu, Hamish Cunningham, and Yorick Wilks. Named entity recognition from diverse text types. In *Recent Advances in Natural Language Processing 2001 Conference*, 2001.

Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the 7th Conference on Natural Language Learning*, 2003.

Nancy McCracken. Combining Techniques for Event Extraction in Summary Reports. In *Proceedings of the Workshop on Event Extraction and Synthesis, held in conjunction with the AAAI Conference*, 2006.

David D. McDonald. Internal and external evidence in the identification and semantic categorization of proper names. *Corpus processing for lexical acquisition*, 1996.

Kathleen McKeown, Regina Barzilay, John Chen, David K. Elson, David K. Evans, Judith Klavans, Ani Nenkova, Barry Schiffman, and Sergey Sigelman. Columbia's newsblaster: New features and future directions. In *Proceedings of NAACL-HLT*, 2003.

Xiangzeng Meng and Lei Liu. On Retrieval of Flash Animations Based on Visual Features. *Technologies for E-Learning and Digital Entertainment*, 2008.

Fien De Meulder, Walter Daelemans, and Veronique Hoste. A named entity recognition system for dutch. *Language and Computers*, 2002.

Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition without gazetteers. In *Proceedings of the 9th Conference on European Chapter of the Association for Computational Linguistics*, 1999.

Miquel Millan, David Sánchez, and Antonio Moreno. Unsupervised Web-based Automatic Annotation. In *Proceedings of the 4th STAIRS Conference: Starting AI Researchers' Symposium*, 2008.

David Miller, Richard Schwartz, Ralph Weischedel, and Rebecca Stone. Named entity extraction from broadcast news. In *Proceedings of the Broadcast News Workshop*, 1999.

David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th Conference on Information and Knowledge Management*, 2008.

Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

Tom M. Mitchell, Justin Betteridge, Andrew Carlson, Estevam R. Hruschka Jr., and Richard C. Wang. Populating the Semantic Web by Macro-Reading Internet Text. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

Nilo Mitra and Yves Lafon. Soap version 1.2 part 0: Primer. Technical report, W3C, 2007.

Dmitry Myagkikh. Anfrage strukturierter Datenbasen mithilfe natürlicher Sprache im Kontext des Question-Answering. Master's thesis, Dresden University of Technology, 2012.

David Nadeau. *Semi-Supervised Named Entity Recognition: Learning to Recognize 100 Entity Types with little Supervision.* PhD thesis, Ottawa-Carleton Institute for Computer Science, 2007.

David Nadeau. What is a Named Entity? Blog Post, 2008. http://yooname.wordpress.com/2008/02/12/what-is-a-named-entity/.

David Nadeau and Satoshi Sekine. A Survey of Named Entity Recognition and Classification. *Named Entities: Recognition, Classification and Use*, 2009.

Martina Naughton, Nicola Stokes, and Joe Carthy. Investigating statistical techniques for sentence-level event classification. In *Proceedings of the 22nd International Conference on Computational Linguistics*, 2008.

Zaiqing Nie, Yunxiao Ma, Shuming Shi, Ji-Rong Wen, and Wei-Ying Ma. Web object retrieval. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.

Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, 2000.

Cheng Niu, Wei Li, Jihong Ding, and Rohini K. Srihari. Bootstrapping for named entity tagging using concept-based seeds. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003.

Joel Nothman. Learning Named Entity Recognition from Wikipedia. Master's thesis, The University of Sydney Australia, 2008.

Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, 2012.

Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. Downloading hidden web content. In *Proceedings of the Joint Conference on Digital Libraries*, 2005.

Christopher Olston and Marc Najork. Web Crawling. *Foundations and Trends in Information Retrieval*, 2010.

STAT OWL. StatOwl.com - Statistical analysis and market research of Internet usage trends. Website, 2012. URL {`http://www.statowl.com/`}.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. Technical report, Stanford InfoLab, 1999.

Sandeep Pandey, Krithi Ramamritham, and Soumen Chakrabarti. Monitoring the Dynamic Web to respond to Continuous Queries. In *Proceedings of the 12th International Conference on World Wide Web*, 2003.

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-Scale Distributional Similarity and Entity Set Expansion. In *Proceeings of EMNLP*, 2009.

Marius Paşca. Acquisition of categorized named entities for web search. In *Proceedings of the 13th International Conference on Information and Knowledge Management*, 2004.

Marius Paşca. Weakly-supervised discovery of named entities using web search queries. In *Proceedings of the 16th Conference on Information and Knowledge Management*, 2007.

Marius Paşca and Benjamin van Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

Jon Patrick, Casey Whitelaw, and Robert Munro. Slinerc: The Sydney language-independent named entity recogniser and classifier. In *Proceedings Conference on Natural Language Learning*, 2002.

Georgios Petasis, Frantz Vichot, Francis Wolinski, Georgios Paliouras, Vangelis Karkaletsis, and Constantine D. Spyropoulos. Using machine learning to maintain rule-based named-entity recognition and classification systems. In *Proceedings of the 39th Meeting on Association for Computational Linguistics*, 2001.

Jakub Piskorski, Hristo Tanev, and Pinar O. Wennerberg. Extracting violent events from on-line news for ontology population. In *Proceedings of the 10th International Conference on Business Information Systems*, 2007.

Jerrey Pound, Peter Mika, and Hugo Zaragoza. Ad-hoc object retrieval in the web of data. In *Proceedings of the 19th International Conference on World Wide Web*, 2010.

Sameer S. Pradhan, Gabriel Illouz, Sasha J. Blair-Goldensohn, Hazen Schlaikjer, Valerie Krugler, Elena Filatova, Pablo A. Duboue, Hong Yu, Rebecca J. Passonneau, Steven Bethard, Valileios Hatzivassiloglou, Wayne Ward, Dan Jurafsky, Kathleen R. McKeown, and James H. Martin. Building a foundation system for producing short answers to factual questions. In *Proceedings of the 11th Text Retrieval Conference*, 2002.

Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Readings in speech recognition*, 1990.

Dragomir Radev, Timothy Allison, Sasha Blair-Goldensohn, John Blitzer, Arda Celebi, Stanko Dimitrov, Elliott Drabek, Ali Hakim, Wai Lam, Danyu Liu, Jahna Otterbacher, Hong Qi, Horacio Saggion, Simone Teufel, Adam Winkel, and Zhu Zhang. Mead - a platform for multidocument multilingual text summarization. In *Proceedings of LREC*, 2004.

Dragomir Radev, Weigua Fan, Hong Qi, Harris Wu, and Amardeep Grewal. Probabilistic question answering on the web. *Journal of the American Society for Information Science and Technology*, 2005.

Anand Rajaraman. Kosmix: Exploring the Deep Web using Taxonomies and Categorization. *IEEE Data Engineering Bulletin*, 2009a.

Anand Rajaraman. Kosmix: high-performance topic exploration using the deep web. *Proceedings of the VLDB Endowment*, 2009b.

Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gn Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation*, 2006.

Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the 13th Conference on Computational Natural Language Learning*, 2009.

Lisa F. Rau. Extracting company names from text. In *Proceedings of the 7th Conference on Artificial Intelligence Applications*, 1991.

Stephanie Reese. Quick Stat: 72.6% of Internet Users Will Buy Online in 2011, 2011. URL `{http://www.emarketer.com/blog/index.php/tag/percent-of-internet-users-who-shop-online/}`.

Sandro Reichert. *Analyse und Vorhersage der Aktualisierungen von Web-Feeds.* PhD thesis, Dresden University of Technology, 2012.

Sandro Reichert, David Urbansky, Klemens Muthmann, Philipp Katz, Matthias Wauer, and Alexander Schill. Feeding the world: a comprehensive dataset and analysis of a real world snapshot of web feeds. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, 2011.

Jason D. Rennie and Tommi Jaakkola. Using term informativeness for named entity detection. In *Proceedings of the 28th International SIGIR Conference on Research and development in Information Retrieval*, 2005.

Ellen Riloff and Rosie Jones. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999.

Daniel E. Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the 13th International Conference on World Wide Web*, 2004.

Ian Rose, Rohan Murty, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, and Matt Welsh. Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *Proceedings of the Symposium on Networked Systems Design and Implementation*, 2007.

Bemjamin Rosenfeld and Ronen Feldman. URES: an unsupervised web relation extraction system. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, 2006.

Marc Rössler. *Korpus-Adaptive Eigennamenerkennung*. PhD thesis, Universität Duisburg-Essen, 2007.

Kugatsu Sadamitsu, Kuniko Saito, Kenji Imamura, and Genichiro Kikui. Entity set expansion using topic information. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

Daniel Sandler, Alan Mislove, Ansley Post, and Peter Druschel. FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, 2005.

Erik F. Sang and Fien De Meulder. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of the 7th Conference on Natural Language Learning*, 2003a.

Erik F. Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *Development*, 2003b.

Luis Sarmento, Valentin B. Jijkoun, Maarten de Rijke, and Eugenio Oliveira. "More like these": growing entity classes from seeds. In *Proceedings of the 16th Conference on Information and Knowledge Management*, 2007.

Nico Schlaefer. Pattern learning and knowledge annotation for question answering, 2005.

Guus Schreiber. OWL: the Web Ontology Language, 2004. URL `{http://www.cs.vu.nl/~guus/public/2004-owl-brisbane/all.htm}`.

Satoshi Sekine. NYU: Description of the Japanese NE System used for MET-2. In *Proceedings of the 7th Message Understanding Conference*, 1998.

Satoshi Sekine and Chikashi Nobata. Definition, dictionaries and tagger for extended named entity hierarchy. In *Proceedings of the Language Resources and Evaluation Conference*, 2004. URL `{http://nlp.cs.nyu.edu/ene/zentaizu6\_1\_2eng.jpg}`.

Purvesh Shah, David Schneider, Cynthia Matuszek, Robert C. Kahlert, Bjorn Aldag, David Baxter, John Cabral, Michael Witbrock, and Jon Curtis. Automated population of Cyc: Extracting Information about Named-Entities from the Web. In *Proceedings of the 19th International FLAIRS Conference*, 2006.

Yu sheng Lai, Kuao ann Fung, and Chung hsien Wu. Faq mining via list detection. In *Proceedings of the International Conference on Computational Linguistics*, 2002.

Ka C. Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient Monitoring Algorithm for Fast News Alerts. *Transactions on Knowledge and Data Engineering*, 2007a.

Ka C. Sia, Junghoo Cho, Koji Hino, Yun Chi, Shenghuo Zhu, and Belle L. Tseng. Monitoring RSS Feeds Based on User Browsing Pattern. In *Proceedings of the International Conference on Weblogs and Social Media*, 2007b.

Steven Soderland, Oren Etzioni, Tal Shaked, and Daniel S. Weld. The use of Web-based statistics to validate information extraction. In *Papers from the AAAI-2004 Workshop on Adaptive Text Extraction and Mining*, 2004.

Michael M. Stark and Richard F. Riesenfeld. Wordnet: An Electronic Lexical Database. In *Proceedings of the 11th Eurographics Workshop on Rendering*, 1998.

Fabian M. Suchanek. *Automated Construction and Growth of a Large Ontology*. PhD thesis, Saarland University, 2009.

Jian Sun, Jianfeng Gao, Lei Zhang, Ming Zhou, and Changning Huang. Chinese named entity identification using class-based language model. In *Proceedings of COLING*, 2002.

György Szarvas, Richard Farkas, and Róbert Ormándi. Improving a state-of-the-art named entity recognition system using the world wide web. *Advances in Data Mining. Theoretical Aspects and Applications*, 2007.

Partha P. Talukdar, Thorsten Brants, and Mark L. Pereira. A Context Pattern Induction Method for Named Entity Extraction. In *Proceedings of the 10th Conference on Computational Natural Language Learning*, 2006.

Qingzhao Tan and Prasenjit Mitra. Clustering-based incremental web crawling. *Transactions on Information Systems*, 2010.

Katsumi Tanaka. Web knowledge extraction for improving search. In *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, 2008.

Hristo Tanev, Jakub Piskorski, and Martin Atkinson. Real-Time News Event Extraction for Global Crisis Monitoring. In *Proceedings of the 13th International Conference on Natural Language and Information Systems*, 2008.

Jayram S. Thathachar, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar Information Extraction System. *IEEE Data Engineering Bulletin*, 2006.

Mai-Vu Tran, Tien-Tung Nguyen, Thanh-Son Nguyen, and Hoang-Quynh Le. Automatic Named Entity Set Expansion Using Semantic Rules and Wrappers for Unary Relations. In *Proceedings of the International Conference on Asian Language Processing*, 2010.

Peter D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, 2001.

Peter D. Turney. Measuring Semantic Similarity by Latent Relational Analysis. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 1992.

Lance Ulanoff. Google Knowledge Graph Could Change Search Forever, 2012. URL `{http://mashable.com/2012/02/13/google-knowledge-graph-change-search/}`.

David Urbansky. WebKnox: Web Knowledge Extraction. Master's thesis, Dresden University of Technology, 2009.

David Urbansky, James A. Thom, and Marius Feldmann. WebKnox: Web Knowledge Extraction. In *Proceedings of the 13th Australasian Document Computing Symposium*, 2008.

David Urbansky, Marius Feldmann, James A. Thom, and Alexander Schill. Entity Extraction from the Web with WebKnox. In *Proceedings of the 6th Atlantic Web Intelligence Conference*, 2009a.

David Urbansky, Daniel Schuster, Maximilian Walther, and Alexander Schill. Focused Question Answer Extraction From Q/A Rich Websites. In *Proceedings of the 8th IADIS International Conference WWW/Internet*, 2009b.

David Urbansky, Robert Willner, and Alexander Schill. Ontofly: Ontology Engineering using the Web. In *Proceedings of the International Conference on Machine and Web Intelligence*, 2010.

David Urbansky, Klemens Muthmann, Philipp Katz, and Sandro Reichert. Palladian: A toolkit for Internet Information Retrieval and Extraction. Website, 2011a. URL `{http://www.palladian.ws/documents/palladianBook.pdf}`.

David Urbansky, Klemens Muthmann, Lars Kreisz, and Alexander Schill. Areca: Online Comparison of Research Results. In *Proceedings of the 5th International Conference on Weblogs and Social Media*, 2011b.

David Urbansky, Sandro Reichert, Klemens Muthmann, Daniel Schuster, and Alexander Schill. An Optimized Web Feed Aggregation Approach for Generic Feed Types. In *Proceedings of the 5th International Conference on Weblogs and Social Media*, 2011c.

David Urbansky, James A. Thom, Daniel Schuster, and Alexander Schill. Entity List Completion using the Semantic Web. In *Proceedings of the 10th International Semantic Web Conference*, 2011d.

David Urbansky, James A. Thom, Daniel Schuster, and Alexander Schill. Training a named entity recognizer on the web. In *Proceedings of the 12th International Conference on Web Information System Engineering*, 2011e.

Dietrich van der Weken, Mike Nachtegael, and Etienne E. Kerre. An overview of similarity measures for images. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2002.

Cornelius J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

Vadim von Brzeski, Utku Irmak, and Reiner Kraft. Leveraging context in user-centric entity detection systems. In *Proceedings of the 16th Conference on Information and Knowledge Management*, 2007.

Vishnu Vyas, Patrick Pantel, and Eric Crestan. Helping editors choose better seed sets for entity set expansion. In *Proceedings of the 18th Conference on Information and Knowledge Management*, 2009.

Richard C. Wang and William W. Cohen. Language-Independent Set Expansion of Named Entities Using the Web. In *IEEE International Conference on Data Mining*, 2007.

Wei Wang, Chuan Xiao, Xuemin Lin, and Chengqi Zhang. Efficient approximate entity extraction with edit distance constraints. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, 2009.

Wei Wang, Dongyan Zhao, Lei Zou, Dong Wang, and Weiguo Zheng. Extracting 5W1H Event Semantic Elements from Chinese Online News. In *WAIM*, 2010.

Yefeng Wang. Annotating and recognising named entities in clinical notes. In *Proceedings of the International Joint Conference on Natural Language Processing: Student Research Workshop*, 2009.

Ziyang Wang. *Incremental Web Search: Tracking Changes in the Web*. PhD thesis, New York University, 2006.

Michael S. Waterman and Temple F. Smith. Identification of common molecular subsequences. *Journal of Molecular Biology*, 1981.

Martin Werner. Suche und Indexierung multimedialer Objekte im Web. Master's thesis, Dresden University of Technology, 2010.

Casey Whitelaw and Jon Patrick. Evaluating corpora for named entity recognition using character-level features. *Lecture notes in computer science*, 2003.

Casey Whitelaw, Alex Kehlenbeck, Nemanja Petrovic, and Lyle Ungar. Web-Scale Named Entity Recognition. In *Proceedings of the 31st International SIGIR Conference on Research and Development in Information Retrieval*, 2008.

Robert Willner. Browserbasierte Erstellung und Erweiterung von Domänenontologien. Master's thesis, Dresden University of Technology, 2010.

Robert Willner. Erzeugung von klassifizierten Vorschlägen für die semi-automatische Erweiterung von Domänenontologien im Browser. Master's thesis, Dresden University of Technology, 2011.

Joel L. Wolf, Mark S. Squillante, Philip S. Yu, Jayachandran Sethuraman, and Leyla Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International Conference on World Wide Web*, 2002.

Dan Wu, Wee Sun Lee, Nan Ye, and Hai Leong Chieu. Domain adaptive bootstrapping for named entity recognition. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

Dekai Wu, Grace Ngai, Marine Carpuat, Jeppe Larsen, and Yongsheng Yang. Boosting for named entity recognition. In *Proceedings of the 6th Conference on Natural Language Learning*, 2002.

Fei Wu and Daniel S. Weld. Automatically refining the wikipedia infobox ontology. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.

Minji Wu and Amelie Marian. Corroborating Answers from Multiple Web Sources. In *Proceedings of the 10th International Workshop on Web and Databases*, 2007.

Martin Wunderwald. NewsX: Event Extraction from News Articles. Master's thesis, Dresden University of Technology, 2011.

Xiao Xu, Weizhe Zhang, Hongli Zhang, and Binxing Fang. Scale-adaptable recrawl strategies for DHT-based distributed web crawling system. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*, 2010.

Sibel Yaman, Dilek Hakkani-Tur, and Gokhan Tur. Combining Semantic and Syntactic Information Sources for 5-W Question Answering. In *Proceedings of the 10zh Conference of the International Speech Communication Association*, 2009.

Jun Yang, Qing Li, Liu Wenyin, and Yueting Zhuang. Searching for flash movies on the Web: A content and context based framework. *World Wide Web*, 2005.

Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st International SIGIR Conference on Research and Development in Information Retrieval*, 1998.

Yiming Yang, Jaime G. Carbonell, Ralf D. Brown, Thomas Pierce, Brian T. Archibald, and Xin Liu. Learning Approaches for Detecting and Tracking News Events. *IEEE Intelligent Systems*, 1999.

Roman Yangarber. Verification of facts across document boundaries. In *Proceedings of the International Workshop on Intelligent Information Access*, 2006.

Alexander Yates. *Information Extraction from the Web: Techniques and Applications.* PhD thesis, University of Washington, 2007.

Alexander Yates, Michael J. Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. TextRunner: Open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2007.

Lei Zhang and Bing Liu. Entity set expansion in opinion documents. In *Proceedings of the 22nd Conference on Hypertext and Hypermedia*, 2011.

Shubin Zhao and Jonathan Betz. Corroborate and Learn Facts from the Web. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

Zhiping Zheng. AnswerBus question answering system. In *Proceedings of the 2nd International Conference on Human Language Technology Research*, 2002.

Xiaohua Zhou, Xiaodan Zhang, and Xiaohua Hu. Dragon Toolkit: Incorporating Auto-learned Semantic Knowledge into Large-Scale Text Retrieval and Mining. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, 2007.