12-19-2008

A Semi-Supervised Information Extraction Framework for Large Redundant Corpora

Eric Normand

University of New Orleans

A Semi-Supervised Information Extraction Framework for Large Redundant Corpora

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Information Assurance

by

Eric Normand

B.G.S. University of New Orleans, 2002

December, 2008

Acknowledgements

Thanks must be extended to the following organizations and individuals:

Naval Research Laboratory
Kevin Shaw
John Sample
For generosity and support.

Mahdi Abdelguerfi
Shengru Tu
Golden Richard III
For guidance.

Bruce Lin
Elias Ioup
For advice and ideas.

My parents
For always being there.

Virginia Medinilla
For her unending patience.

Contents

List of Figures

Abstract

The vast majority of text freely available on the Internet is not available in a form that computers can understand. There have been numerous approaches to automatically extract information from human-readable sources. The most successful attempts rely on vast training sets of data. Others have succeeded in extracting restricted subsets of the available information. These approaches have limited use and require domain knowledge to be coded into the application.

The current thesis proposes a novel framework for Information Extraction. From large sets of documents, the system develops statistical models of the data the user wishes to query which generally avoid the limitations and complexity of most Information Extractions systems. The framework uses a semi-supervised approach to minimize human input. It also eliminates the need for external Named Entity Recognition systems by relying on freely available databases. The final result is a query-answering system which extracts information from large corpora with a high degree of accuracy.

Keywords: Information Extraction, Natural Language Processing, Support Vector Machine, Machine Learning, Information Retrieval, unstructured text

Chapter 1

Introduction

The amount of information available on the World Wide Web was estimated in 2003 at 167 terabytes [27]. This number is triple what was available in 2000, when a previous study was performed. The annual amount of email generated in 2003 worldwide was 667,585 terabytes. There is obviously a vast amount of text information available in digital form.

There are many documents that are about the same topic on the Internet, and therefore contain very similar information using different wordings. Their information can be compared to verify the correctness of the source. The redundancy is an asset that can be exploited.

While the amount of text being produced is continually increasing, so is the amount of computing power available to process it. The large majority of these computers are dedicated to indexing for a human to search through.

Further, there is a growing awareness of the richness of the text information available. Thousands of people contribute to Wikipedia, a free encyclopedia, creating millions of entries. There is a strong desire to make this information available not only to the 1.4 billion Internet users [12], but to make it available for entry into databases, question-answering systems, and Artificial Intelligence efforts.

The efforts to make the large amount of textual information available in a structured, machine-usable form have been largely successful. However, these efforts, in general, have focused on domain-dependant approaches which do not transfer well to the problem in the large.

Presented here is a framework on which a semi-supervised Information Extraction system can be built. The framework itself is relatively dynamic and flexible. It can be easily adapted by the end user to a wide variety of domains with minimal effort. A system built on this framework will extract highly structured data from large, unstructured text sources as typically found on the World Wide Web.

The current research borrows from work in the literature in a number of ways. Firstly, the current work relies on and takes advantage of the redundancy of the information contained in large text databases. Textual sources can be aggregated into large databases. The accuracy of the algorithm increases with the size of the database. This idea is used in Etzioni et al. [15, 16] and Agichtein and Gravano [1].

Secondly, the current work approaches the problem from a typical point of view. Instead of modeling the task as a textual understanding problem, the system models the problem as one of statistical correlation. In essence, the system attempts to correlate syntactical and lexical features with semantic meaning. The semantic meaning is inputed by the user in the form of a query. It is therefore a query-answering system as opposed to a free-form extraction engine.

Finally, a semi-supervised algorithm is used to reduce human input. Several papers [1, 3, 14, 33] use a bootstrapping algorithm which trains a system with a minimal set of seed values instead of the vast number of training examples needed in a supervised learning method.

Although the current research owes a great debt to prior work, the system presented here diverges in three important ways. Firstly, the current work does not depend on any one domain (as do the majority of previous research; see Etzioni et al. [15] and Brin [3] for exceptions). Extending the system to work in new contexts is trivially simple and requires minimal domain expertise. The system is also not dependant on a

particular representation of the sentence nor on the choice of classifier. Most prior work relies heavily on these two choices.

Secondly, most systems that generate statistical models throw out features that correlate with undesirable information [1, 6, 23, 33, 36, 37]. However, those features can help filter out information. The current system uses patterns that correlate well with desirable data and patterns that correlate with undesirable data to classify unknown values into the appropriate category.

Lastly, the system does not depend on sophisticated, statistical Named Entity Recognition systems (see Section 1.4). The current Named Entity Recognition systems have limited domains or require large, tagged training sets to extend their domain. The current system uses a simplified, lexical Named Entity Extraction step that is simple to extend or modify.

There is a growing need to give programmatic access to the vast amount of information provided in unstructured text on the Internet. The current work grows out of lessons learned from prior research, while pursuing a novel approach to the task. The system described in this paper is a practical query-answering system that can extract information from large document sets in a domain-independent way. A demonstration system is developed that can accurately extract a variety of different kinds of information from Wikipedia.

1.1 Information Extraction

This section will review the field of Information Extraction, its goals, and its relationship to other fields. By the end of this chapter, the reader should have a feeling for the problems Information Extraction attempts to solve and the potential issues that arise.

It is helpful to rely on a metaphor to explain the concept of Information Extraction. In describing the problem and challenge of Information Extraction, this text will rely on the metaphor of reading and understanding a completely foreign language. The reader might imagine text in a language as alien to him or her as possible, such as Icelandic or Mandarin.

When given a sentence in Icelandic, a person unfamiliar with the language would have a difficult time trying to understand it. The text is written in an unfamiliar script, with words and grammar that are incomprehensible. One could imagine the difficulty the person would have to identify all of the place names in the sentence. This is a typical task asked of Information Extraction systems. The challenge of Information Extraction is to create an automated system that can identify specific information that is contained in that sentence or texts like it.

The definition found in Moens [30] describes the field more precisely.

> Information extraction is the identification, and consequent or concurrent classification and structuring into semantic classes, of specific information found in unstructured data sources, such as natural language text, making the information more suitable for information processing tasks.

Information Extraction seeks to use computers to automatically apply structure to unstructured information. In the case of the current research, the unstructured information used is textual. Other possible sources are audio, image, and video files [30].

The definition of the goal is left purposefully general. Any kind of structural information can be applied to the unstructured source. Not only is Information Extraction applied to extract content from text, it is also applied to extract meta-data such as authoring information (who wrote it, etc).

Some typical examples of the kinds of structure applied to text are:

- Names of people, places, and organizations
- Relationship between two people
- Geographic relationships (X is near Y)
- Dates of an event

The distinguishing characteristic between structured and unstructured data is that unstructured text is *computationally opaque* [30]. That is, a computer cannot immediately interpret and make use of the data, as one could with a database or XML file. Information Extraction creates structured information which can be added to a database or queried in a structured way.

1.2 Data Mining

Data mining is often used in Information Extraction tasks. Data mining seeks to find patterns and relationships in data [38]. The methods developed in data mining include classifiers and statistical modeling techniques.

Information Extraction tasks often (but not always) include a data mining step. Unstructured information textual information is converted to some structured, numerical form. That structured data is then processed using standard data mining techniques, such as clustering, probabilistic networks, and statistical classifiers.

The data mining methods used to train statistical models include *supervised*, *unsupervised*, and *semi-supervised*.

Supervised training requires a training set that is completely labeled—that is, given a set of data points, each data point in the set is correctly classified into the appropriate class. The computer then learns to classify other points in the same way.

Unsupervised training uses properties of the data alone to find patterns. No other information is given to the system. The system often finds new, unexpected patterns (like dividing the set into more classes than a human would). The typical data mining technique in this category is clustering, where data points are grouped based solely on similarity.

Semi-supervised learning is where only a (usually small) portion of the training data points are classified into their categories. One could choose to ignore all of the unclassified data—leaving only a small number of classified points. However, one is often able to improve the performance of the classification technique by incorporating the unclassified data into the training phase.

1.3 Natural Language Processing

As will become clear in section 2, many approaches (including the current work) use Natural Language Processing techniques. Natural Language Processing studies the understanding and generation of human languages. The language understanding portion is what is mainly used in Information Extraction.

Three general subtasks of Natural Language Processing are often used in Information Extraction [18]. They are used because the information they generate can be useful to the pattern matchers and classifiers used in Information Extraction.

When one is presented with a sentence in Icelandic, one might like to know which words were nouns, which were verbs, and which were prepositions. That information could help one decipher the sentence. The task of identifying the part of speech of words in a sentence is called *Part of speech tagging*. This information can be useful for subsequent tasks because it provides more information than the sequence of characters that make up the words. See Figure 1.1 for an example.

In English, some things are described with multiple words. One common examples is the "World Health Organization". Instead of interpreting those words separately, it could be helpful to group them together for the purposes of interpretation. *Noun phrase identification* is another Natural Language Processing method which serves to group nouns into noun phrases. Some Information Extraction methods treat noun phrase groups in the same way they treat nouns.

Given a sentence in a foreign language, it could be useful to know which words were the subjects of which verbs, what words are modified by adjectives, and so on. This information is gleaned from the text through the action called *deep parsing*. Parsing is the task of grammatically deconstructing a sentence to understand its syntactic structure. Parsing generates a tree or a graph from a sentence. It is thought by some that a parse tree reveals more useful information than representing it as a series of words. However, parsing at a

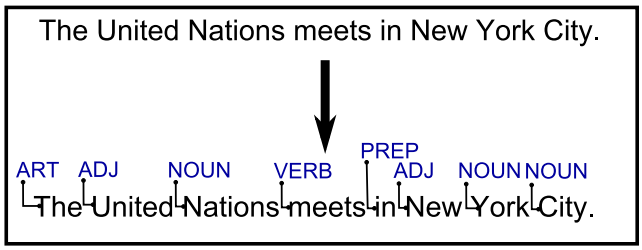

Figure 1.1: The result of Part of Speech tagging on a sample sentence.

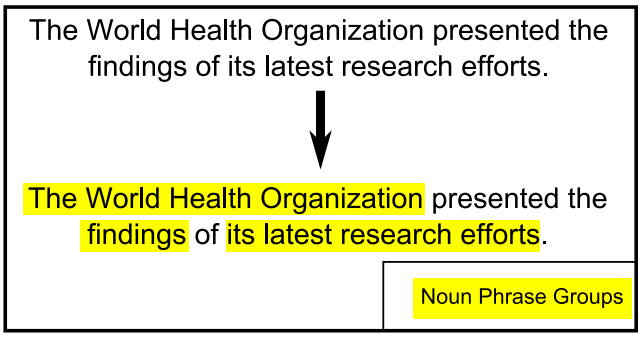

Figure 1.2: An example Noun Phrase Grouping. Highlighting represent noun phrases.

high accuracy can be computationally expensive. Figure 1.3 shows a syntax tree of a sentence. Figure 1.4 shows a dependency parse, which shows dependency relationships between words and phrases.

These three methods are all considered to be forms of parsing (with varying computational costs).

1.4 Common Information Extraction Tasks

The Message Understanding Conference (MUC) has essentially defined the tasks that Information Extraction attempts to perform [19].

Named Entity Recognition is the task of identifying and classifying the words associated with "entities". In the MUC sponsored tasks, the entities were classified as one of *Person, Location*, and *Organization*. The task occasionally included *Time/Date* and *Money*. Named Entity Recognition is often performed as a preprocessing step of other Information Extraction tasks. It is also sometimes used instead of a parsing step. Figure 1.5 gives an example of the result of Named Entity Recognition.

Relation Extraction is the identification and classification of the relation between two entities in a text. The current work addresses this task. The fundamental concept of this task is that much information that is pertinent to the structure of a database deals with how the entities interact and relate. It is therefore fruitful to extract information pertaining to the relations between entities.

Relations can include anything from familial relations to geographic relations ("Paris is the capital of France" relates Paris to France; "Russia is near China" relates those two countries). Relation Extraction attempts to determine what the nature of the relation is from textual evidence.

Coreference resolution seeks to identify cases where different words refer to the same entity or concept. This occurs in several cases: when the same name is used in two different places to refer to the same entity; when two different names refer to the same entity; when pronouns are used; when anaphoric phrases are used (such as in the two sentences "I took a trip to Paris. When I arrived in the city, I found a hotel"; "the city" refers to the same thing as "Paris"). This task can also inform later extraction tasks.

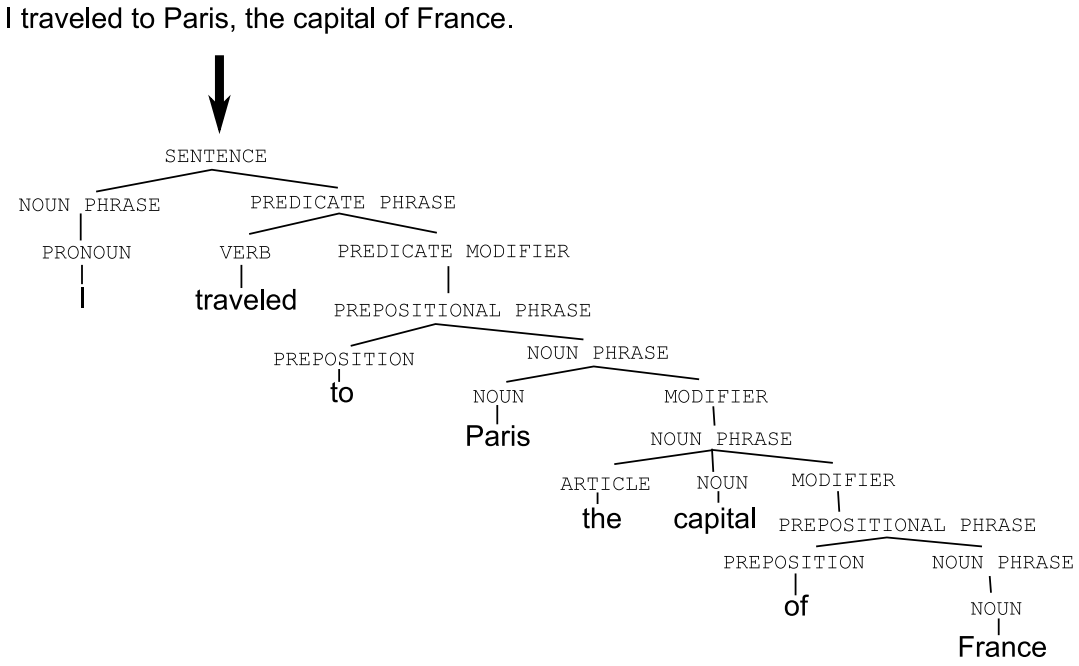

Figure 1.3: A syntax tree.

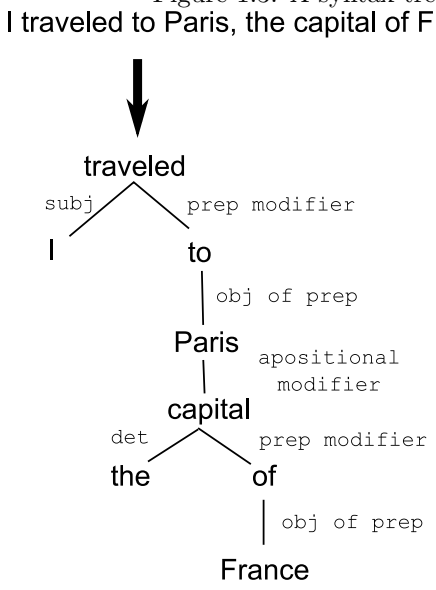

Figure 1.4: An example dependency parse. Note the labeled edges.

1.5 Evaluation Metrics

There are three standard metrics for evaluating an Information Extraction task. They are borrowed from Information Retrieval [40].

Recall measures the proportion of target facts in a text that are correctly extracted. Recall is a measure of completeness without attention to the incorrectly extracted facts.

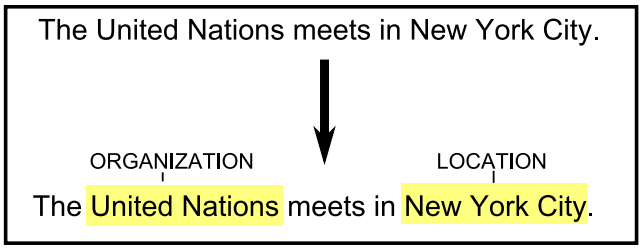

Figure 1.5: An example result Named Entity Recognition task.

$$\text{Recall} = \frac{|\{\text{extracted facts}\} \cap \{\text{true facts}\}|}{|\{\text{true facts}\}|} \tag{1.1}$$

where the truth value of the fact is domain dependent. The fact can be compared to a database of known facts or can be hand-tagged in the document.

Precision measures the proportion of extracted facts that are correct. Precision is a measure of correctness.

$$\text{Precision} = \frac{|\{\text{extracted facts}\} \cap \{\text{true facts}\}|}{|\{\text{extracted facts}\}|} \tag{1.2}$$

F-measure combines Recall and Precision into a single metric. The F-measure is a weighted harmonic mean of Recall and Precision. It allows for a single metric that weights Recall and Precision by their relative importance to the task at hand.

$$F_\beta = \frac{(1+\beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{1.3}$$

where β is the relative importance of Recall over Precision. F_1 weights Recall equally to Precision. F_2 weights Recall twice as important as Precision. $F_{0.5}$ weights Precision twice as important as Recall.

These metrics all fall in the range $[0, 1]$. This makes them useful to compare evaluations across algorithms and implementations. There is, however, discrepancies in how the notion of truth is calculated. In general, supervised training methods consider the annotations in the training and testing documents as truth. There is therefore an explicitly defined set of true facts—those that occur in the annotations. In semi-supervised methods, where no (or very few) annotations are available, other methods of determining truth are required [1].

When no annotations exist, it is possible to evaluate one's methods over a domain with known truth values. The truth value of a fact can then be checked using that knowledge. The question then becomes whether it is reasonable to expect the system to extract facts that are not available in the text. A measure of what is extractable is then required.

The common method for evaluating a domain where no complete model of truth can be used is simply to count the number of returned facts. Two algorithms can then compare the size of the dataset returned.

Chapter 2

Prior Works

This chapter organizes the related work in the field along three axes. Each axis forms a continuum along which the techniques fall. The majority of research involved in evaluating different classifiers and their parameters applied as different feature sets. The triadic taxonomy done here will encompass the majority of prior research.

2.1 Training

The primary axis determines the type of training the system uses. The type of training that a research approach uses is one of the ways the techniques differ from each other. The type training also determines to a large extent how much human work is required to train the system.

Training techniques can range from fully hand-written to totally unsupervised methods. Figure 2.1 shows an illustration of the tradeoffs made when choosing a training method.

Hand-written techniques are the most time consuming. People must analyze hundreds of documents looking for patterns that will be useful for extracting the information. Not only is it expensive—it is also error-prone, inaccurate, and tends to miss a large portion of patterns. Some approaches to hand-written techniques do show promise, such as Fundel et al. [17] and Daraselia et al. [11], but they are still the most time-consuming and inflexible.

Unsupervised methods require no human input to train the system. This can take the form of automatically determining different groupings of data. Since no human input is required, the system can be applied to many different domains without the expense of developing a new training corpus for each one. However, these systems must deal with a grave difficulty: the data they get do not necessarily correspond to the user's idea of relevance. The systems have no human guide to learn from [38].

Between the two extremes are *supervised* and *semi-supervised methods* (sometimes called weakly supervised methods). *Supervised methods* generally require a large training corpus that is completely annotated by hand by a domain expert. The system uses machine learning techniques to mimic the annotation choices made by humans. It can be rather costly, but usually gives very accurate results. Often the results can take into account the subtleties of human discernment.

Semi-supervised methods try to gain the advantages of human knowledge without the intensive labor of a completely annotated corpus. These methods typically require only a fraction of the annotated data given to supervised methods. Others provide a handful of examples of the kind of data one wishes to extract. The disadvantage is that they are less accurate than supervised methods.

2.2 Depth of Parsing

The second axis is the depth of parsing performed on the sentences (see Section 1.3). A deeper parse is computationally more expensive. Many algorithms have been derived using only part of speech tagging or

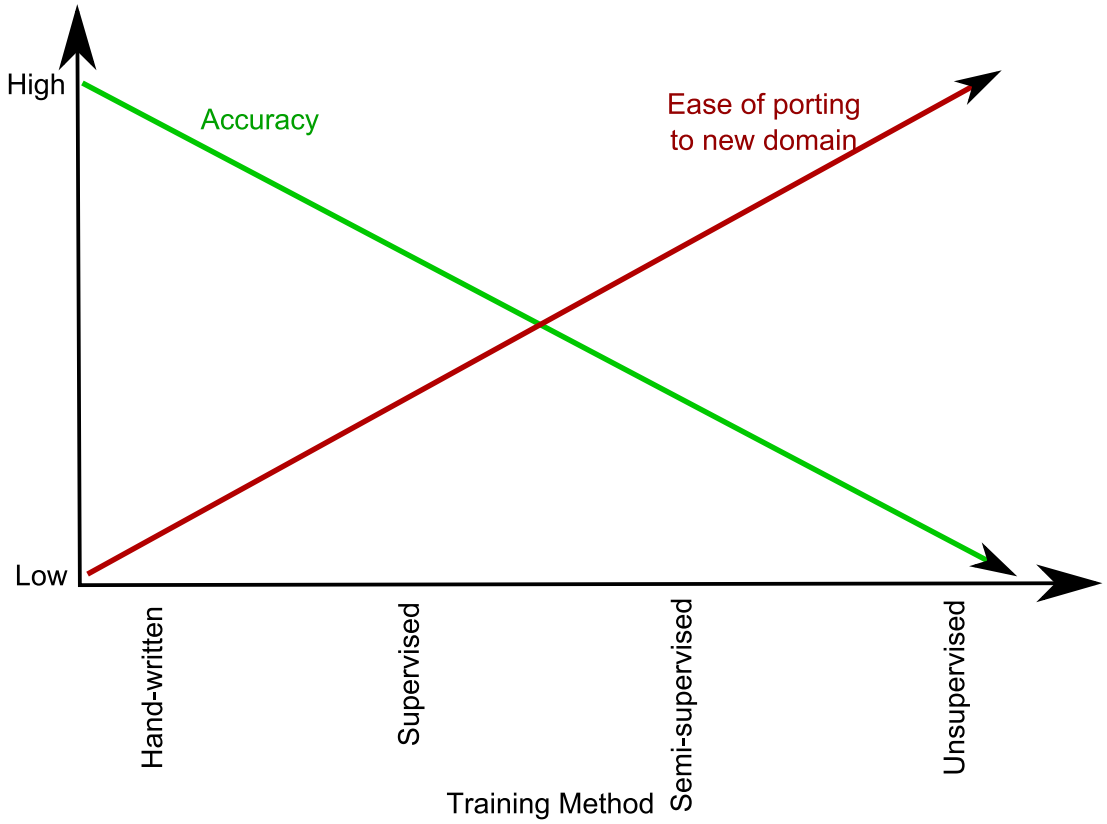

Figure 2.1: In general, training methods that result in higher accuracy require more human effort to port to a new domain.

noun phrase clustering. Some systems use a Named Entity Recognizer as the only parse. Named Entity Recognition can also be performed in addition to another form of parsing. Still others perform no parsing at all and rely only on textual matching (such as Grishman [18]).

In general, the more computationally expensive an operation, the more information can be gleaned from the sentence. Figure 2.2 visualizes this point.

2.3 Machine Learning Technique

The third axis is how the system learns to generalize from a training set. This axis can range from rote techniques to advanced statistical machine learning.

Figure 2.3 illustrates the compromise necessary when choosing among the following classification techniques.

2.3.1 Rote techniques

Rote techniques use a deterministic predicate to extract the information. A simple example is a regular expression. If the regular expression matches a sentence, then the information contained in the groupings (parentheses) of the regular expression form the entities in the data value. In general, rote techniques use one or more patterns that match the parsed representation of the sentence. If at least one pattern matches,

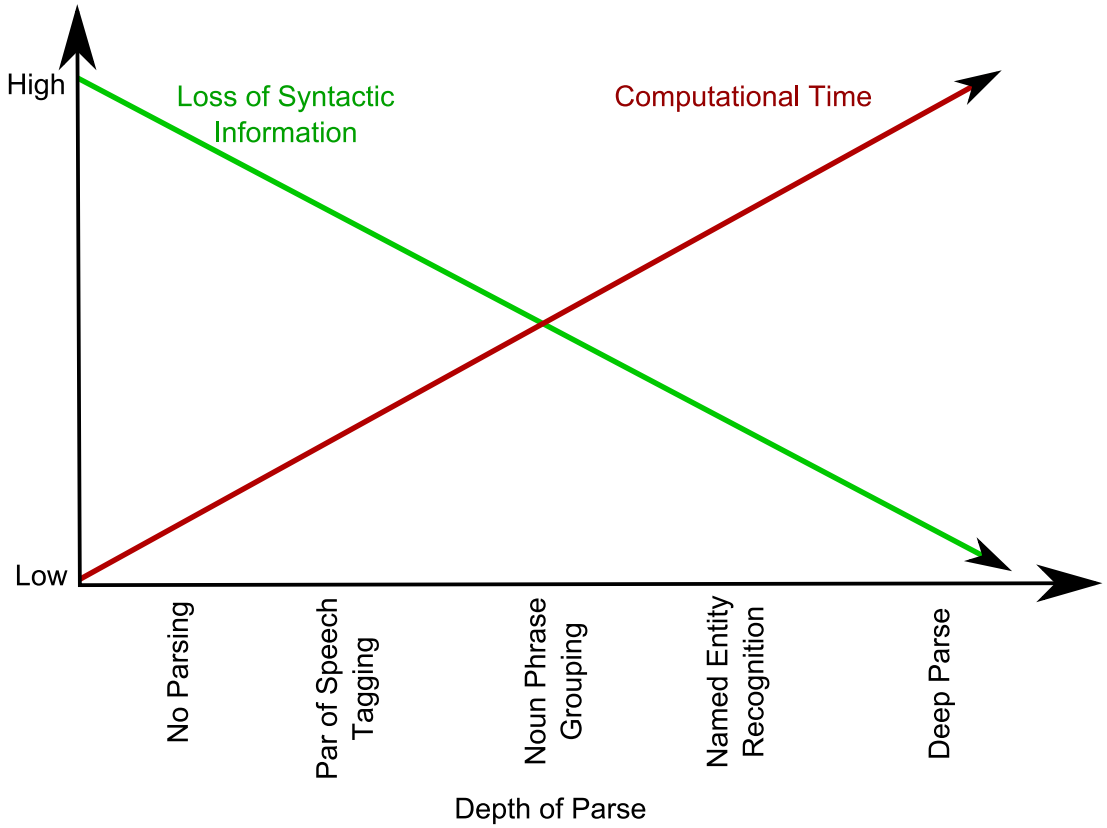

Figure 2.2: Investing more computational time to parsing leads to higher syntactic information gains.

the data in the sentence is considered relevant and the system extracts it. If no pattern matches, the data is not extracted.

The rote patterns are typically generated in one of two ways. The first way is *exact matching*. This approach is where each training instance generates a single pattern. Those patterns are not modified, though they can be kept or discarded (see Etzioni et al. [15] for an example).

The second way is called *generalization*. This is where some simple form of machine learning is used to make the generated patterns more general, and therefore more resistant to the presence of noise (Califf and Mooney [6], Huffman [23], and Soderland [35] use this method).

2.3.2 Information retrieval methods

In *information retrieval methods*, metrics of document relevance are used to measure the relevance of individual sequences of words (example: Agichtein and Gravano [1]).

2.3.3 Statistical techniques

Statistical techniques are typified by the use of many training points. They rely on statistical averaging or probability to find patterns in the data.

Probabilistic networks are trained statistical methods that label sequences of data [8]. The sentence can be treated as a sequence of words. The network then predicts labels for the words that correspond to the roles in the relation that is being extracted [25]. For instance, in the relation *CapitalCityOf*, Paris could

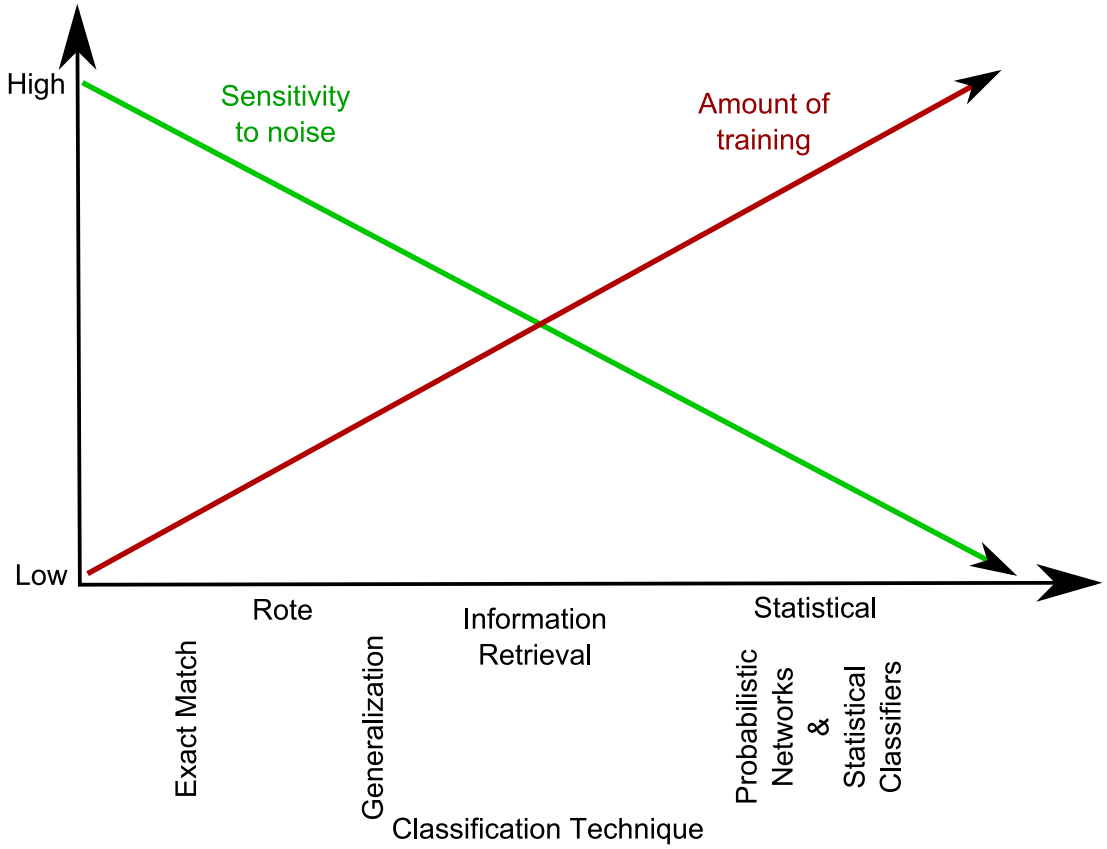

Figure 2.3: Techniques that are less sensitive to noise require more training.

be labeled as *CITY* and France could be labeled as *COUNTRY*. One can then extract the words with a meaningful label. Examples of probabilistic networks are Hidden Markov Models and Conditional Random Fields.

Statistical classifiers use data mining techniques. Sentences or facts can be modelled as feature vectors appropriate for the classifier chosen. The classifier can be trained using traditional supervised, semi-supervised, or unsupervised approaches. Examples of statistical classifiers are Naive Bayes and Support Vector Machines. Examples include Culotta and Sorensen [10], Jie and Min [24], and Bunescu and Mooney [5].

2.4 Thought Experiment

This chapter also describes the problem definition each paper attempts to solve. There is a wide variety of problem definitions, and their explanations can get quite tedious. To make it easier for the reader, a thought experiment has been devised. It will make use of the same metaphor used previously which was about deciphering a foreign language.

The thought experiment is called the *Icelandic Library*. It goes as follows. A person, named Bob, is in a library in Iceland. To simplify matters, all books are written in Icelandic—and there are no pictures. The person does not speak Icelandic. Bob is given a task, which is defined differently with each research paper description. Bob is to use the resources of the library, which can differ with each prior research, to perform the task. It should be understood that though the explanation for each problem is stated as if it were in Icelandic, all papers in fact operate on English language text.

2.5 Literature Review

The following sections divide the literature into four categories along the axis of type of training. An explanation of where each prior work falls in the remaining two axes is detailed in its description.

2.5.1 Hand-written methods

The hand-written methods generally consist of people writing regular expression-like patterns to match certain patterns in text. The patterns are typically very specific to a few syntactic patterns. These kinds of methods do not offer very much help for changing domains.

The following papers are classified as hand-written methods because they require a large, domain-specific portion to be hand-coded.

Fundel et al. [17] was built to extract protein-protein interactions from biological paper abstracts. In this paper, Bob's task in the Icelandic Library to go through every book in the library, sentence by sentence, and create a list of protein-protein interactions, cataloguing the type of interaction. Each sentence in the library is already parsed into a dependency tree (an example of deep parsing. See Section 1.3) for Bob to use. Bob also has a dictionary of protein names (in Icelandic) to guide the search.

The paper proposes the following solution. After finding a sentence with a protein name from the dictionary, Bob will perform certain simple transformations to the dependency tree. These transformations include building noun phrases and reconstructing lists from their syntactic interpretation. Bob then looks for three syntactic patterns in the resulting tree. These patterns correspond to what the authors claim are the three most important syntactic forms for expressing relations. Quoting the text:

> Currently, we use three rules that reflect the constructs that are most frequently used in English language for describing relations, namely:
>
> 1. effector-relation-effectee ('A activates B').
> 2. relation-of-effectee-by-effector ('Activation of A by B').
> 3. relation-between-effector-and-effectee ('Interaction between A and B').

The triplet <effector, relation, effectee >is extracted if it matches one of the patterns above. Each of the patterns is hard coded (an example of a rote technique. See Section 1.3). These patterns are fixed at design-time—and they are chosen based on the language patterns of biological abstracts. It is easy to see that extending this system to new domains would necessarily require adding a new dictionary (to replace the protein dictionary) and to reanalyze the new text sources for possible syntactic patterns that correspond with the semantic meaning that is desired.

Daraselia et al. [11] uses a hand-generated ontology of protein-protein interaction. An ontology, in this sense, is a collection of data structures describing how to translate between an input sentence and another data structure called a *frame*. A frame is just a template with slots. The slots define admissible values. Synonyms, keywords, and protein names are all defined in the ontology.

In this problem, Bob's task is to choose a form (called a *frame* in the paper) from a list of possible forms based on rules in the ontology. Then each form is filled out based on the same rules. Bob is again given a dependency parse for each sentence. The rules are of the type "use the form called *Inhibition* when the verb of the sentence is 'inhibits"'.

The paper analyzes the example sentence "P53 inhibits apoptosis." The system performs a dependency parse of the sentence, then applies the ontology to the resulting tree.

The word "inhibits" is in the ontology, and activates a frame. The frame is called "Inhibition" and describe what useful information the system should look for in terms of chemical inhibition. The "Inhibition" frame has an "agent" slot and a "patient" slot. The "agent" slot needs a protein name that is the subject of the verb. A search through the ontology reveals that "P53" is a protein name, and fills the requirements for "agent" in this frame. "P53" is inserted into the agent slot. The "patient" slot needs a direct object that is a cell process. "apoptosis" is a cell process and a direct object of "inhibits" and therefore fulfils the

requirements for "patient". The frame has been filled in successfully and can be stored in a database. The tree is walked this way, until there is no more information to be extracted. This can be considered a rote process since no machine learning is performed.

The construction of the ontology is very time-consuming. It is constructed by hand by domain experts. And the ontology is domain-specific. Little or none of the domain knowledge encoded in the ontology can be transfered to another application domain. For instance, in addition to the complex network of frames and other concepts, the ontology includes 6,000 words and their meanings in the ontology. These words are domain specific—words not meaningful to the search were not included. It would be very expensive to generate a dictionary of that size for every domain one would like to query. Although this technique is very powerful, the expense is prohibitive.

2.5.2 Supervised methods

The Message Understanding Conferences (MUC's) encourage supervised models of the problem domain [19]. The corpora used in its proceedings are fully annotated. The domain is completely specified ahead of time. This makes it easy to apply traditional machine learning techniques. The focus is placed on choosing an appropriate feature set.

The main choices that must be made in a supervised model are the choice of feature set and the choice of classifier.

Rote methods

The simplest classifier is a pattern matcher. It is a simple, deterministic function that classifies the type of relation present in a sentence.

Huffman [23] presents a system that creates a dictionary of patterns from training examples that can be later used to extract information from other documents. In this paper, Bob is given the task of generating tree patterns that indicate a certain semantic meaning and can be applied to new sentences.

Bob is given a book of sentences that are parsed into a syntax tree. In each sentence, two words are marked as different from the others. These are the two entities in the relation. From this book, Bob must generate a small set of patterns that will be able to match entities that have the same relationship as the words in the book.

The solution presented in the paper is that Bob should determine the path along the parse trees that are important to the relation using a set of rules. The rules specify which words to ignore and which to keep. These paths are written down, with the marked words replaced by blanks. When all of the sentences in the book have been processed, Bob is to compare all of the patterns to each other, finding patterns that have similar structure. When two patterns are similar, words from one pattern that do not match the corresponding word in the other pattern are replaced by a wild-card and that new pattern replaces the other two. In this way, the patterns are generalized

The list of patterns can then be applied to new sentences parsed into a syntax tree. If any other patterns match, the words that correspond to blanks in the pattern are returned as extracted information.

Transferring this system to new domains is better than with the hand-written methods. In this case, one needs only create a new training set (the book of marked sentences given to Bob). However, with small training sets, the system can be inaccurate. If only one sentence exists in the training set, only one pattern can be generated. A sentence must match it exactly to be extracted. Considering how many ways there are to phrase the same fact, it would take many training examples to generate a system with many generalized patterns to result in high Recall.

Califf and Mooney [6] has a similar system. It was designed to extract information from job postings on Usenet. Bob is given one form to fill out that contain the attributes of job postings—including job title, company, and salary. Bob is also given a set of postings in Icelandic and their corresponding filled-in form. From this, Bob must generate a small set of patterns that can be applied to new job postings to automatically fill in the forms. The job postings are given as-is—that is, no parsing is performed.

The solution is to have Bob write out the words surrounding the words from each blank in the form as a pattern. When all of the training examples have been done this way, the patterns corresponding to the same blanks are compared. Bob chooses two patterns at random, finds the similarities between them, and generalizes the differences so that they will match the same set of sentences. The generalizations are fixed. One example is to generalize "house" and "cup" to match any noun. The new generalized pattern is then matched over the training set, extracting values. If it extracts any wrong values, the generalization is discarded. If it does not match any wrong values, Bob iterates through generalizations until no more generalizations can be performed that produce only correct extractions.

The generalized patterns are added back into the list, replacing the two patterns they were generalized from. When there are no more possible generalizations, the list of patterns is returned.

Generalizations are performed in order to avoid over-fitting patterns to the training data. This process lets the set of patterns remain small and also allows the system to deal with noise in the training set.

Although this method is more powerful than Huffman [23], it still requires an expert to create a training set for each new domain. It is therefore time consuming and inflexible.

Califf et al. [7] extends the system from Califf and Mooney [6] to allow for generalizations from Wordnet classes. For example, combining patterns for the words "house" and "apartment" could lead to a pattern that matches any word in the "dwelling" category.

Soderland [35] uses a simple pattern matcher similar to regular expressions. The way it creates patterns is not significantly different from other pattern matching systems. Bob is given the task of generating patterns that will match a given relation between entities. In this research paper, Bob is allowed to ask a librarian to read sentences and identify the relationship stated in the sentence. This lets Bob build a new training set.

The solution the paper proposes is one that tries to minimize the number of sentences that are required of the librarian. It interactively asks the user to tag sentences. The pattern generation method is not materially different from the previously described methods. The system selects the sentences the user will label based on three categories: Instances covered by an existing pattern—to check the patterns, Instances not covered by a pattern but covered by a generalization of a pattern—to confirm that a generalization is correct, and Instances not covered by any pattern—to expand the number of patterns.

By randomly training and retraining from these three classes of sentence, the user slowly increases the number of training instances available to the system. This process is called *active learning*. It uses feedback to minimize the number of training instances required. The drawback is that the fundamental means of learning has not changed, so only a small measure of efficiency is gained.

Statistical methods

Support Vector Machines are a state of the art statistical classifier. The kernel function is essentially a measure of similarity. Much research is being done on the choice of features and the kernel function. Research is on-going into new kernel functions that can act on trees—dependency parse trees specifically.

Bunescu and Mooney [4] builds a system that uses a kernel tailored for subgraphs of a dependency tree. Bob is given a task of classifying the relationship between two given entities in a sentence into one of a set of classes. He has at his disposal a set of dependency trees from sentences and the categories they should be classified into.

The solution the authors propose is for Bob to find all sentences with the same training class. Bob then must extract the shortest path in the tree from one entity to the next (since it works only on binary relations). Bob can compare two sentences based on the number of words the shortest paths have in common.

An SVM is trained with this data. A new sentence is classified by extracting the shortest path between entities and classifying it with the SVM. The authors make the hypothesis that the most important syntactic and lexical features of a relation occur along the shortest path between the two entities on the dependency tree. The shortest path is augmented with part of speech and Wordnet hypernym information.

Bunescu and Mooney [5] develops a kernel function that applies to sequences of words. It is equivalent to the previous work except that it applies to word subsequences instead of dependency graphs. The subsequence kernel counts the number of matching subsequences.

The target domain is protein-protein interactions in biological abstracts. The length of the sequences is limited to four words. The insight in this paper is that only three common expressions of relationship are considered. By restricting the feature set, they both speed up the algorithm and look only at very compelling data.

Since the sequences are no longer than four, this is computationally inexpensive. Using a subsequence kernel allows for a shallow parse or no parse at all instead of an expensive deep parse.

Culotta and Sorensen [10] develops a kernel function that operates on the dependency graph. The hypothesis laid out is that the syntactic relation between two named entities is local—that it is expressed in the shortest subtree of the dependency parse tree containing both entities. This hypothesis is not as strong as the thesis proposed in Bunescu and Mooney [4]. The kernel recursively counts the number of common subtrees two trees share. The dependency tree is augmented with many features—including the part of speech and Wordnet hypernyms.

The entities of interest are identified in the sentence by using a Named Entity Recognition phase. This technique allows for a fairly robust identification of names, places, and organizations. However, the Named Entity Recognition system used and those in common use are limited in what types of entities they identify. They also typically require supervised learning, so they are costly to extend to new domains.

Harabagiu et al. [20] augments the work in Culotta and Sorensen [10] by adding new features to the dependency graph. An ontology of one million words was added to provide information about what noun phrases are the arguments of which predicates.

Jie and Min [24] uses a Support Vector Machine to classify extracted relations. It performs noun-phrase chunking instead of doing a deep parse. It argues (and its results confirm) that little extra precision is gained from doing a full dependency parse. Its feature set is not different from the other work on SVM's.

Liu et al. [26] considers a set of syntactic features to combine into a feature vector for input to a kernel. The feature extractions are hand-coded. The syntactic features they develop include patterns such as "parent node of protein name" in the parse tree. This is important because it tries to combine syntactic features into a feature vector to be used in a classifier. Unfortunately, the syntactic features are selected by hand specifically for the domain. This means that the features are likely not to work in a new domain.

The typical statistical approach is to find a sentence that has the entities in question and to try to extract their relationship from the sentence. Culotta et al. [8] turns that around and assumes that the text is about some entity—the *subject* of the text. The goal, then, is to find the relationships between the subject and the other entities mentioned in the text. In this approach, Bob is given a sentence and the subject of the document from which it is taken. Bob then is asked to find the relationship of the entities in the sentence and the subject.

This approach models *context* in a simple way in which implicit mentions of the subject of a text can be resolved to an explicit entity. For instance, in an article about Germany, one might find the sentence "The capital is Berlin." No where in the sentence is Germany mentioned. It is implied that Germany should be understood because the entire document is about Germany. Most approaches miss these references. Section 6.3 discusses a way to augment the current work with a similar kind of context.

Culotta et al. [8] trains a Conditional Random Field (See Section 2.3) to label the words in a sentence. The labels can be either IRRELEVANT or the name of the class. For instance, in an article about George W. Bush, the sentence "He is married to Laura Bush." would label the name "Laura Bush" as WIFE and the rest of the words as IRRELEVANT. It is then trivial to extract out the information.

Another approach is taken in Xiao et al. [41]. The classification is done using a maximum entropy model. Features include the words surrounding the entities. It performs a dependency parse, and uses features from those trees.

The Supervised methods follow basically the same approach: use a standard classifier on a feature vector derived from the sentence. The choice of classifier and the feature set are thus very important and form the bulk of the research.

2.5.3 Unsupervised methods

Unsupervised methods do not require human input. They typically categorize or cluster data automatically using standard clustering techniques.

Etzioni et al. [15] describes a system called "KnowItAll" that uses the World Wide Web as a giant corpus of sentences (the library in the Icelandic Library metaphor). The user can ask Bob to give a list of entities that are of a certain type, or to give a list of pairs of entities that are in a certain relationship.

The authors have developed a set of domain-independent extraction rules. These rules are not meant to exhaustively cover every way an English sentence can describe a relation. The rules instead leverage the sheer size of the Internet to "play the averages". The principle is that on average, there is a sentence that does match a certain rule somewhere on the Internet that contains the desired information.

The following are some example patterns. NP means noun phrase and {} means the surrounded text is optional.

- NP1 {","} "such as" NPList2
- NP1 {","} "and other" NP2
- NP1 {","} "including" NPList2
- NP1 "is a" NP2
- NP1 "is the" NP2 "of" NP3
- "the" NP1 "of" NP2 "is" NP3

The user can query KnowItAll for "cities in France". It will then search various search engines with the queries "cities in France such as", "and other cities in France" and "is a city in France". The resulting pages are scraped for the sentence containing the text. The patterns are applied and the resulting information is extracted.

This system is interesting for three reasons. Firstly, it is more active than the normal approach. It seeks the information that it knows how to extract instead of learning how to extract information that is given to it.

Secondly, it uses the redundancy of the size of the dataset (effectively, the entire searchable Internet) to make up for its lack of intelligence.

Lastly, it allows for a wide range of domains. The only human input needed is the expression of the query.

The KnowItAll system is classified here as shallow parsing and rote. It is considered unsupervised since it requires no human input to extend its domain.

Etzioni et al. [16] extends the KnowItAll system. Three unsupervised methods are added. The *Rule Learning* system augments the domain-independent rules from the original KnowItAll system by extracting text patterns using seeds generated by the domain-independent rules. This could be considered a case of semi-supervised learning (i.e., learning from a small number of examples) but the seed examples are automatically generated by the system itself. It has been classified here because there is no human input required.

The second extension to KnowItAll is *Subclass Extraction*. The reasoning behind the system is simple. If the user wants to query the system for a list of scientists, the original KnowItAll would use patterns equivalent to "scientists such as" to find sentences that contain scientists. However, the system would miss sentences that match "physicists such as". The Subclass Extraction system uses KnowItAll to extract a list of highly probable subclass names to augment the query. The system finds new subclass names by querying itself for "types of scientist".

The third extension is called *List Extraction*. In List Extraction, the system queries a search engine for one of the many structured lists of items on the Internet. The items are chosen at random from the lists generated by the original KnowItAll system. The goal is to find many structured lists containing the items and correlate the results together. Many queries are made, and the items are ranked by the number of lists they appear in.

This method is interesting because it shows that a high precision algorithm can be used as an initial stage to generate high-precision seed candidates. Those seeds can then be fed to higher recall stages.

It also shows how useful redundancy can be. Instead of trying to deduce what a single sentence is saying, the system uses the redundant nature of multiple documents to extract and verify the information.

Hasegawa et al. [21] introduces a system that clusters pairs of entities together based on similar contexts. The algorithm works as follows: given a set of documents, a shallow parse is performed. The system identifies pairs of entities that are near each other. The words that come between the pairs in the sentences are stored. The system then clusters the pairs of entities based on the similarities of the set of intervening words.

After the clustering is performed, the most common words from each cluster are chosen as the cluster label. The results indicated that this word was accurate for clusters of at least a certain size.

Shinyama and Sekine [34] develops a system for automatically classifying relations among entities. Their system works on an entire document (as opposed to a sentence). It identifies entities in the document. The system records the context words surrounding each entity. It then assumes that a relationship exists between each pair of entities in the document by reasoning that if they did not have a relationship, they wouldn't be in the same document. The system performs this analysis on multiple documents, keeping the entities separate.

The system then performs clustering on the pairs of entities across multiple documents. The goal is to use the context surrounding each word to cluster entities with similar "roles" together. For instance, in an article about Hurricane Katrina, the entity "New Orleans" might be surrounded by "was hit". "Katrina" would be surrounded by "headed" as in "Katrina headed North". In another document about the tsunami that ravaged Myanmar, "Myanmar" also is found near "was hit". "Tsunami" contains the context "headed" as well. These two pairs of entities would be clustered near each other, indicating the same basic relation.

Although this clustering approach was not taken in the current research, Shinyama and Sekine [34] is interesting because it looks at a document as a whole. Most systems rely on the information contained in one sentence. It has the possibility to extract information other systems miss.

Unsupervised methods are still cutting-edge. They do not perform as well as supervised methods. However, there is much promise that they will be able to recognize patterns and relations that humans cannot.

2.5.4 Semi-supervised methods

Semi-supervised methods usually involve applying machine learning to a set of seed values. These values are a handful of examples, as opposed to an exhaustively tagged document set. These examples are then used with an untagged document set to train the machine learning classifier. Very often, the classifier is then applied to the documents to extract more values. These values are added to the seed set, and the algorithm is iterated. By doing this, the seed set "grows" to a large enough size to achieve a high recall. If the seeds are chosen well, high precision is possible as well.

The advantage of this approach is that only a few examples are needed from a human. The training document set can be as large as one wants—no extra human input is required. It has the advantage of generalizing well to many different domains. A domain is defined by its examples. But it also avoids the disadvantage of unsupervised methods in that the categories are defined by the person giving the examples.

Riloff [33] develops a system that extracts relations from text. It requires only texts classified as relevant or irrelevant to the domain. Bob is tasked with identifying sentences that specify a particular relation among the entities in the sentence. At his disposal, Bob has a set of documents that he knows are relevant to the relation at hand.

The solution is simple. Every potential entity is extracted from the document, including the surrounding text. That text is made into a pattern. The patterns are then evaluated: they are rated as the ratio of the number of times that pattern occurs in a relevant document over how many times it occurs in any document. A confidence metric is then calculated from the matching patterns. Entries with high confidence are considered relevant.

The system generates thousands of patterns, so only the highest ranked patterns are maintained. This technique is interesting because it does not require the same amount of work a supervised system would

need. Instead, one only needs to indicate which documents are relevant to the domain. However, the task of classifying and reclassifying documents for each domain would become tedious.

Brin [3] describes a program that uses the scale of the World Wide Web to extract information. Bob is asked to find other pairs of entities that are in the same relation as a few examples. He can only use the text of the books in the library for his search.

The solution the author proposes is to search for the examples in the books and record the interleaving text between them. Then, that interleaving text is searched for in the books. The words on either end of the text (beginning and end) are assumed to be correct values. They are added to the seed set and the system iterates.

There were several design decisions in this system. The first was that it was unimportant to have a high recall since the scale of the Internet made even a poor recall system return many thousands of values. Secondly, the precision was of utmost importance since bad seeds could cause the entire set to become polluted with more bad seeds. The system therefore made no attempt at generalizing the patterns—it relied only on the scale of the Web to generate enough values.

Unfortunately, this system had no automated method for making sure few bad seeds got into the seed set. A manual "cleaning" stage was necessary to avoid very bad values.

Agichtein and Gravano [1] introduces a system called SnowBall. It follows the basic semi-supervised model outlined above. Given a few seed values and a set of training documents, the system develops a set of patterns.

The patterns match using an information retrieval algorithm. Specifically, it uses a method commonly used to compare documents. The pattern is a weighted bag of words of all the words occurring around the two entities in a sentence. The number of common words is summed up, weighted by the weight of each word. This lets a score be calculated for how related a pattern is to a piece of text.

An implied negative set is created. The domain is assumed to be "many-to-one" or "one-to-one". For instance, the concept of which country is in which continent is many-to-one. A country can only be in one continent and many countries are in a continent. The idea of capital cities is often one-to-one. A capital belongs to one country, and the country belongs to one capital.

Because there is an assumed unique component (the country in the country-continent relation above), a negative set is created. For instance, if "Paris is the capital of France" is a known value, it is implied that no other city is the capital of France. This allows the seed set to be much larger than what the user enters by hand.

The patterns are rated by how many positive and negative seeds they match in the text. This rating is called the *confidence*. Patterns with high confidence match a high proportion of positive seeds and a low proportion of negative seeds.

New values are generated by these patterns from the documents. The values are rated based on the confidence of the patterns that match them. The highest rated values are then fed back into the seed set and the entire algorithm is iterated.

The algorithm showed good results. There are several disadvantages to this approach, though. Firstly, it assumes that a negative set can be created. This is not true in domains that are "many-to-many". Many-to-many domains do not have, in general, any easily calculable negative set. The range of many-to-many domains is very large. Imagine the domain of what countries are allied together.

Secondly, the algorithm relies on a Named Entity Recognition step. While the authors claim that they have used a very reliable and accurate NER system, the problem still remains of training that system. The system they use categorizes entities into three broad categories: LOCATION, ORGANIZATION, and PERSON. The paper also claims that its system can be trained for other categories. However, one of the main advantages of a semi-supervised model is the need for very little human input during training. If the NER system needs to be trained for every domain, then the advantage is lost.

SnowBall does present interesting ideas. First of all, it combines the results of multiple patterns over many sentences. It relies on the idea of data redundancy in multiple documents to achieve higher accuracy. Secondly, it presents a very clear bootstrapping technique. Both of these ideas are essential to the current work.

Agichtein et al. [2] improves on the SnowBall system. It changes the pattern matching algorithm. The original SnowBall system worked on a bag-of-words model of the words surrounding the entities. The improvement is to take word order into account in the pattern matcher. The results of the two systems are then combined to achieve better performance.

The interesting part of this is that SnowBall gives a framework to work in: the system only changed in one part—the pattern matching algorithm. This shows that the overall bootstrapping algorithm is sound and extensible.

Culotta and McCallum [9] creates a system to apply active learning techniques to the semi-supervised approach. It takes the normal active learning approach of choosing the most important unknown values to classify. However, it also weighs into the ranking a measure of the difficulty of classification. This allows the system to more effectively reduce the human input necessary to achieve a desired level of precision.

Mann and Yarowsky [28] combines three different classifiers to match patterns. It uses a Rote system (basic string pattern matching), Naive Bayes (bag of words), and a Conditional Random Field. Combining the results of all of them (using a voting system) increased the accuracy significantly compared to using only one.

The system bootstraps not only the seed set but also the document set. After generating the patterns from a small corpus of tagged documents, it performs a web search of the seed values. The resulting documents are then used to test the patterns and generate new values.

Stevenson and Greenwood [36] presents an interesting approach. It follows the basic bootstrapping method described above. However, instead of adding new values to the seed set, it adds new patterns to a pattern set. It compares the patterns "semantically" to existing, reliable patterns. The patterns are compared based on Wordnet categories.

Pennacchiotti and Pantel [32] describes a system called Espresso. The system differs from the standard bootstrapping algorithm by an adaptation to insufficient data. In small document sets, it queries the web to expand the number of documents available. It also develops metrics for evaluating a patterns and the generated values. These are helpful for filtering the seed set and pattern dictionary.

Pantel and Pennacchiotti [31] extends Espresso by treating patterns with high recall differently from patterns with high precision. Their results are combined differently, giving patterns with high precision the role of evaluating the high-recall patterns. This allows for a higher recall with a small loss in precision.

Suchanek et al. [37] builds a system called LEILA. LEILA parses sentences using a link grammar. A link grammar builds links between pairs of words, similar to a dependency graph. The resulting graph can be traversed as a sequence of words (nodes). The patterns LEILA generates use the path between the two entities in the binary relation. LEILA is restricted to binary relations.

LEILA generates a set of patterns from a seed set of values and an untagged document set. Patterns are then used to generate values. Any patterns that produce values that are in the negative seed set are discarded. The remaining patterns are fed to a classifier, which is trained to identify good patterns.

This system avoids the problem of feature explosion. Since it trains a classifier to identify good patterns, it does not need to compare each sentence with hundreds of generated patterns. It is different from other approaches which aim to train a classifier with interleaving text ([4, 10]) in that it first filters the interleaving text to make sure it does not generate known negative values.

Turney [39] describes a novel approach to relation extraction. The system, at heart, operates like this: given two pairs of entities and a document set, the system discovers all of the intervening text between the pairs in the documents. The system finds the pattern that is shared the most between them. The idea is that what is important is what common relation they share.

The system answers analogy problems like those on the SAT. The approach is novel because it takes the opposite approach to what is commonly taken. Instead of looking for a pattern that will extract a value from a sentence, it compares the patterns generated by two values. In other words, the system asks the question "what is the relationship between the relationship of these values?" For small domains (such as multiple choice analogy problems), this approach has a lot of merit. Many questions could be formulated in this way. For instance "Is Paris the capital of France?" could be formulated as "Does Paris have the same relation to France as Berlin has to Germany'?".

Paper	Training Method	Depth of Parsing	Machine Learning Technique
Fundel et al. [17]	Hand-Written	Deep	Exact Matching
Daraselia et al. [11]	Hand-Written	Deep	Exact Matching
Huffman [23]	Supervised	Deep	Generalization
Califf and Mooney [6]	Supervised	None	Generalization
Soderland [35]	Supervised	None	Generalization
Bunescu and Mooney [4]	Supervised	Deep	Support Vector Machine
Bunescu and Mooney [5]	Supervised	None	Support Vector Machine
Culotta and Sorensen [10]	Supervised	Deep	Support Vector Machine
Jie and Min [24]	Supervised	Noun-phrase Chunking	Support Vector Machine
Liu et al. [26]	Supervised	Deep	Support Vector Machine
Culotta et al. [8]	Supervised	None	Conditional Random Field
Xiao et al. [41]	Supervised	Deep	Maximum Entropy
Riloff [33]	Semi-supervised	Part of Speech	Bayesian Combination
Brin [3]	Semi-supervised	None	Exact Matching
Agichtein and Gravano [1]	Semi-supervised	Named Entity Recognition	Information Retrieval and Bayesian Combination
Culotta and McCallum [9]	Semi-supervised	None	Conditional Random Field
Mann and Yarowsky [28]	Semi-supervised	None	Exact, Naive Bayes, and Conditional Random Field
Stevenson and Greenwood [36]	Semi-supervised	None	Information Retrieval
Pennacchiotti and Pantel [32]	Semi-supervised	Part of Speech	Bayesian Combination
Suchanek et al. [37]	Semi-supervised	Deep	k-Nearest Neighbor and Support Vector Machine
Turney [39]	Semi-supervised	None	k-Nearest Neighbor
Etzioni et al. [15]	Unsupervised	Shallow	Exact Matching
Hasegawa et al. [21]	Unsupervised	Shallow	Clustering

Figure 2.4: The prior works broken down according to the three axes.

Chapter 3

Approach

This chapter deals with the problem of extracting text. The various decisions that have to be made will be explained. The reasoning behind each decision will be detailed. And a list of advantages and disadvantages will be made.

The following are system requirements interpreted from practical usage requirements and the limitations of the systems presented in Chapter 2.

1. Many specific kinds of relations need to be extracted. The relations could be unary, binary, or n-ary. The extraction framework cannot be limited to the order of the relation.

2. The relations are often unbounded. This means that the size of the set of facts corresponding to that relation is unknown by anyone.

3. The relations will be used as specific fields in a database. They are therefore somewhat well-defined. They must match an actual concept that is trying to be captured. Precision is more important than recall since the extracted data will be inserted into a database to be used by other systems as known facts.

4. The relations are not specified ahead of time. They need to be able to be added and modified by end users. This implies that the human effort required to define a new relation is minimized.

5. The domains of the relations (types of entities) are not specified ahead of time. New entity types may be added later. No domain-specific knowledge can be defined in the algorithm.

6. Large, untagged sets of documents are available.

7. The choice of classifier should be unimportant to the functioning of the framework. New classification techniques should be used as they become available. Research is going on into new data mining techniques at a fast rate. The system should be able to incorporate the new research.

8. The type and level of parsing should not factor into the framework. They should be considered implementation details. A review of the literature indicates contentious ideas involving the benefits of parsing [10, 24]. The best practices in parsing have yet to be determined and are likely to change.

3.1 The Three Major Axes

Each of the three axes described in Section 2 is a major decision point. These decisions must be made regardless of the problem formulation. Type of training will be analyzed first.

Hand-written techniques allow for very well-defined domains 2.5.1. Any level of detail is available. It depends only on the amount of effort and time available. Hand-written techniques work well for untagged documents, since they do not require a statistical training stage.

Hand-written methods, however, are largely disused in more recent works. This could be due to the fact that they are extremely time consuming. They do not extend well to new domains. Daraselia et al. [11] builds a large, hand-written ontology. This case is exceptional, however, since the perceived value of extracting very high-quality information exceeds the cost of generating the ontology. The domain of the ontology is very large. However, hand-written methods do not meet the requirements enumerated above. Hand-written methods do not minimize the amount of effort required to add new relations. They also require the domain to be known ahead of time.

Supervised techniques have achieved very high Precision in the literature. This is their strong point. They allow for relations to be well-defined. They can take into account as much subtlety of meaning as can be encoded in the annotations of the documents.

Supervised methods, by definition, require much human input, usually in the form of hand-tagged text. Either the domain of the relations must be known at the outset of training, or the text must be tagged anew for each domain. This, too, does not meet the system's requirements. The tagging of documents requires experts in the domain. Those experts must also be trained to tag the documents consistently—which requires a different kind of expertise. Supervised methods are therefore ruled out because they require an excess of human input.

Unsupervised methods do not require the human input of hand-written and supervised methods. They can adapt to new domains easily.

However, unsupervised methods cannot be relied on to create categories of relations that correspond to a specific category in a database. This works for new domain discovery, but not for extracting a specific kind of information, such as in response to a query from the user. Unsupervised methods will not meet the requirements.

Semi-supervised methods strike a compromise between supervised and unsupervised methods. They require domain knowledge. However, the required amount and precision of the domain knowledge is very low. Often the required human input is trivial. Semi-supervised approaches rely on labeled examples and a large set of unlabeled points [14]. The examples are very natural for humans to provide and large document sets are free on the Internet. They allow users to tell the system "give me more facts like these".

By providing examples, the user is also defining a well-understood concept. The facts extracted can thus be placed into a database. Because the human input is minimal, adding new relations is trivial. A semi-supervised technique has been chosen as the best way to achieve the stated goals for the system.

The next decision is level of parsing. As stated in section 1.3, parsing can range from deep parsing (full syntax or dependency tree) to part of speech (POS) tagging to no parsing at all.

There are disadvantages to deep parsing. Besides the cost of parsing, there is a dependence on the parser. Every parser is different, with differing strengths and advantages. The parser used is usually trained from a corpus. The most common corpus is the Penn TreeBank [29], that has a large set of sentences with their corresponding parse trees. The training parse trees were hand-generated by expert grammarians.

The information gained by deep parsing seems to subsume all of the other levels of parsing. It contains information that is highly correlated to Part Of Speech tagging and noun phrase chunking [18]. The question therefore is one of cost versus benefit.

It is assumed in Jie and Min [24] that by throwing more data at the problem, deep parsing is unnecessary. That is, as the number of training points goes to infinity, the difference in accuracy between a deep-parse solution and a no-parse solution goes to 0.

That being said, there is not infinite data. In fact, at the low number of points (around a few thousand), there is in fact a large benefit from having a fully parsed solution.

Parsing is a relatively expensive operation. Depending on the parser, a sentence could take up to 3 seconds to parse. This is not trivial, especially when considering the sheer number of sentences needed for statistical classification methods. However, there are several mitigating factors:

1. Parallel

   Each sentence can be parsed independently of the others. This allows for massive parallelization of the parsing operation.

2. Cacheable

   For a deterministic parser, the parse for a sentence will always be the same. This means that a given sentence will only need to be parsed once—no matter how many classifiers it passes through.

3. Easily Filtered

   A simple dynamic programming optimization can be performed. An $O(n)$ scan of the sentence can determine if the sentence is worth parsing. If no value for the particular query one is looking for can be extracted from the sentence, it is not worth parsing.

Given these three properties of the system, deep parsing is the best option. Culotta and Sorensen [10] and Bunescu and Mooney [4] indicate that the dependency tree is the most fruitful form of parsing.

The framework of the current system does not rely on the level of parsing. The system is defined in a general sense. It is perhaps not necessary to perform any parsing at all. The evaluation of the usefulness of parsing is left as a subject of future research.

The final decision that must be made is the type of classifier used. Since one of the stated requirements is that the system does not depend on any one specific kind of classifier, it is necessary to model the problem in a way that can be classified by a large set of classifiers.

The Icelandic Library will serve as a metaphor for presenting a clear description of the problem.

Bob is in the Icelandic Library, but he is alone. He is given a sheet of paper.

On the left side of the paper, in a column, one word per line, are many Icelandic words. The person does not understand them. On the right side, there is another list of words, one word per line, in Icelandic. The words appear as no more than a cryptic sequence of characters.

There are lines connecting each of the first five words on the left to words on the right. No pattern is discernible. One line leaves from every word on the left. Some words on the right have two lines connecting to them. Some words on the left have two lines connecting to them. The lines are examples that are to be used in the task. See Figure 3.1.

The task is to draw lines from the remaining words on the left to the appropriate word on the right, based on the same relationship given in the lines already on the page. Any and all of the books in the library can be used.

This paper proposes that Bob proceeds as follows. Bob picks a pair of words connected by lines. He searches through all of the books in the library, looking for sentences that contain that pair of words. He notes the shortest path on the dependency tree of the sentence in a notebook. He does this for each of the example pairs. He then searches through the books again, this time looking for sentences that match those patterns and who also have pairs of words from the sheet of paper. He notes which patterns matched each pair. This information can then be used to determine which lines to draw. How this information is used makes the current research novel.

This model is very similar to that used in the semi-supervised methods in [1, 3, 28, 31–33, 36, 37]. However, it differs in that the system does not look at one small portion of text at a time. It attempts to model the information contained in the entire document set.

The current research proposes that a feature vector be constructed for each candidate fact to be extracted. Each feature in the feature vector will represent a single pattern matching at least one sentence from the document set. Each pattern represents a syntactic-lexical feature of the entities. The classifier is used to determine a relationship between those syntactic-lexical features and semantic meaning. The system will use the standard bootstrapping method to iteratively build a set of values from a small initial set of examples. The approach is described more rigorously in Chapter 4.

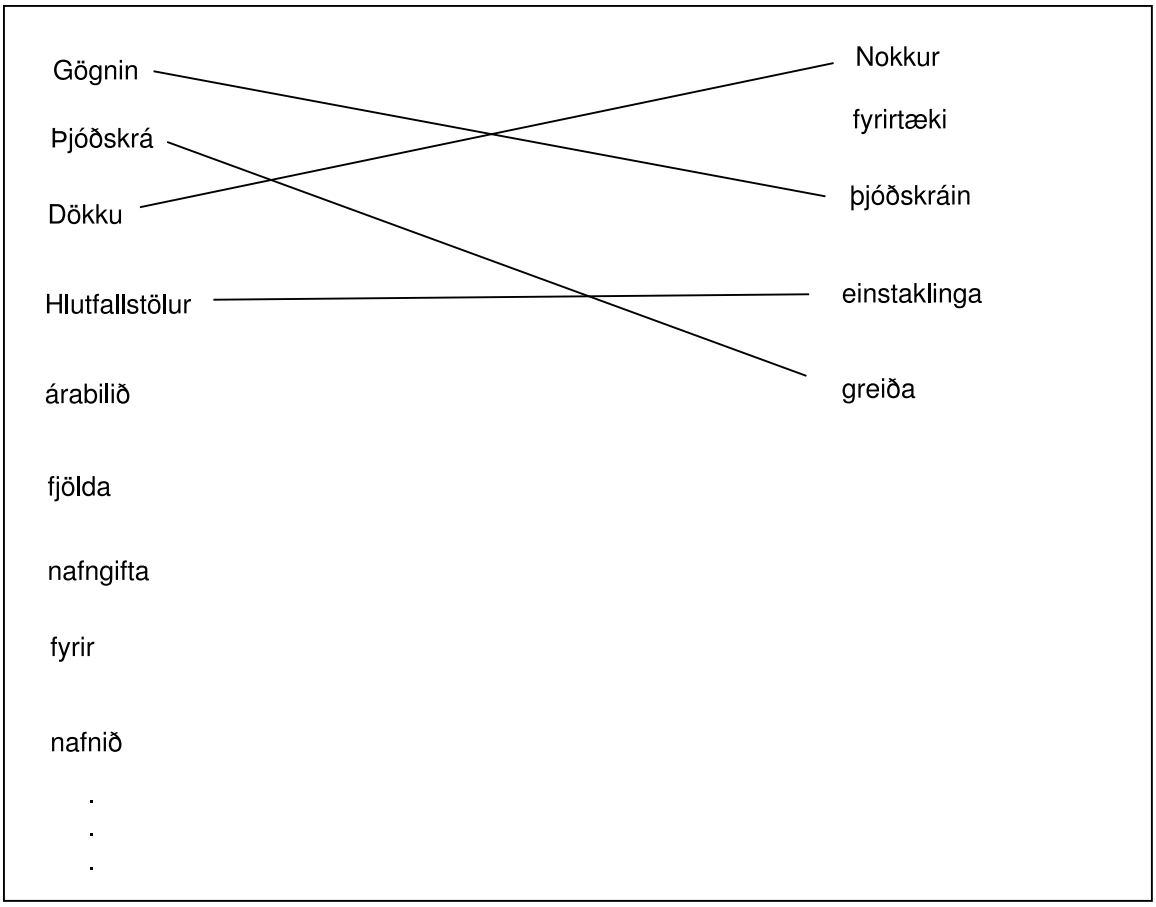

Figure 3.1: A query task definition given to Bob in the Icelandic library. The task is to connect the remaining words on the left with the words on the right in the same way as the examples.

3.2 Comments on the Approach

3.2.1 Choice of classifier

The choice of classifier ranges over many possibilities. In this report, Naive Bayes and Support Vector machines are evaluated. However, most classifiers can be trained on vectors of numeric features. The framework, therefore, remains general.

The Bayesian approach has some advantages (despite requiring some very strong assumptions). First, Bayesian prediction does not require 100% knowledge. Bayesian prediction works on estimates of the various parameters in Bayes' formula. These estimates can be made with incomplete knowledge (see section 5.3).

The second reason the Bayesian approach was chosen was that it accumulates. Adding a new training data point is a constant-time operation, as is reclassifying a point after a new training point is added. It is quite simple to formulate a Bayesian classifier that operates over long periods of time. As more documents are read containing the information to extract, the estimate of the probability of class membership becomes more accurate. This is important in long-running databases, and in cases where the relation can change over time.

Support Vector Machines have a high tolerance for noise and show strong resilience when facing complex feature vectors. Support Vector Machines are expected to perform very well [38].

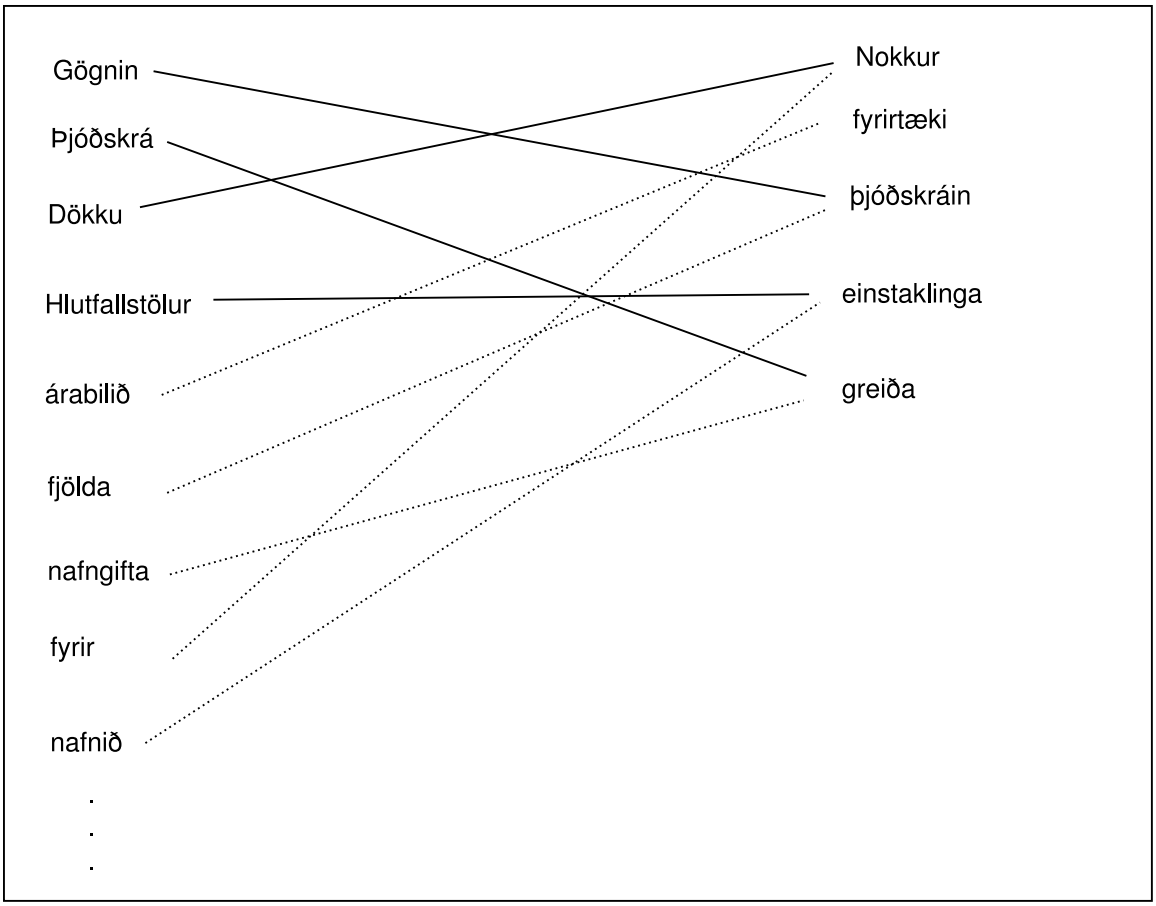

Figure 3.2: The result of Bob completing the task. Dotted lines are those drawn by Bob.

The three principle decisions have been made: Semi-supervised learning using dependency graph and a statistical classifier (Naive Bayes and Support Vector Machine).

3.2.2 Use of domain knowledge

No domain knowledge is needed by the system besides what is given in the query. The possible entities in the relationships are those listed on the page (in the metaphor). This is in contrast to most other research which use a Named Entity Recognition system that is previously trained with domain knowledge, a static list of possible entities (in the case of protein-protein interaction papers), or no system at all to limit the types of entities returned.

The advantage of specifying the domain completely (as in the current system) are many. The first advantage is that the query limits the search space to a large degree—leading to more refined results than those returned by systems that do not limit the space.

The second advantage is that the domains are easily specified and are usually specified already in database form. Many organizations already have a database of the kinds of information that is important to them. These databases can be reused in the queries. Also, if the information is to be added to a database anyway, the lists of entities in the domain probably already exist.

The drawback to the system is that it must be specified completely by a human. The Named Entity Recognition systems in use today primarily use machine learning to identify entities that were not conceived of by a human. However, those machine learning techniques require much more human input than a complete

specification of the domain. The current approach is a balance between the human input and expressive power.

3.2.3 Linguistic pitfalls

Many linguistic pitfalls exist, including ambiguous names and the idea of negation. Ambiguous names exist—for instance "Paris" names a city in France and a city in Texas. This phenomenon is not resolved in the current work. It is assumed that the structure of the query will, on statistical average, eliminate the ambiguity. For instance, if one wants to identify the capital cities of countries, it is unlikely that a sentence talking about Paris, Texas will also talk about a country in the same way a sentence will talk about Paris, France.

Negation is also not addressed. Negation is a problem because a sentence that says "Paris is the capital of France" differs from a sentence that says " Paris is not the capital of France" by one word. But that one word makes the whole sentence mean exactly the opposite of what one would think it would mean based on similarity of the sentences. This problem is usually dealt with either by throwing out negative sentences [17] or by including negation as a feature in the patterns [4].

The reason for neglecting negation is that it was desired to minimize the complexity of the model. Evaluating the importance of dealing with the negation problem is outside of the scope of the current research.

Pronouns and anaphoric phrases are not attempted to be resolved. The low accuracy of the current coreference resolution solutions does not indicate that they would add value to the system.

No context is taken into account. Each pattern matches individual sentences, with no accounting for ideas that surround the sentence in the text. It is hypothesized that this identifying context is not strictly necessary, though simple models of context might make for fruitful future work.

Chapter 4

Formal Methods

This chapter presents a formalized model of the problem. It is this model that will form the framework for actually developing a solution. The definitions in this section will be referred to in later sections.

4.1 Definitions

4.1.1 Tuples

Data must be represented in a structured way in the computer. This section therefore defines a flexible data structure that will be used for the known information and for the unknown information that is to be extracted.

A *tuple* is defined as an ordered series of strings of characters. The choice of character set is arbitrary to the algorithm and should be chosen to accord with the data being analyzed.

$$t_k \equiv \langle t_{k1}, t_{k2}, \cdots, t_{kn} \rangle \tag{4.1}$$

Tuples represent relational data. That is, the strings in the tuple are in some relationship to each other by the very act of being together in a tuple. The relationship might not be what is being looked for, or it could be exactly what is being looked for.

The strings that make up the tuples each themselves represent the name of something. Resolving exactly what each name refers to is its own research project. The problem will not be addressed here.

For instance, the system treats the string "Paris" which in one sentence refers to the capital of France as equivalent to the string "Paris" that refers to the city in Texas. They are treated equivalently because there is no simple way to disambiguate their meaning. It is hypothesized—and later confirmed (see section 5)—that on average, the system performs well even under that kind of ambiguity.

It must be stressed that tuples can contain ambiguity. The following is an example:

1. President Bush flew from Paris to his ranch in Texas.
2. A flash flood struck the city of Paris, Texas early Friday morning.

The tuple $\langle Paris, Texas \rangle$ can be extracted from each of these sentences. Though the meaning of each is different, the system has no privilege to such knowledge. They are considered the same tuple.

The basic unit of data is now defined. The tuple will be the input to queries. It will also be the output of queries.

4.1.2 Classes

The number of possible tuples makes it practically difficult to perform searches over all of them. A notion of *classes* is defined to help focus the search. Classes refer to the type of entity a string names.

Classes are defined in a domain- and implementation-dependant way. If one wants to define a domain of geographic entities, one might classify strings into **Country**, **City**, **Mountain**, etc.

Every string can belong to multiple classes. For instance, the string "France" refers to the country or to many cities by the same name around the world. It would therefore be a member of the class **Country** and the class **City**. We can refer to the set of classes of a string *str* as $class_{str}$.

To formalize classes more concretely, we can define an inclusion function that is a predicate indicating whether a string belongs to the class.

$$\text{country}(str) \equiv \begin{cases} true & \text{if } str \text{ belongs to the class } \textbf{Country}, \\ false & \text{otherwise.} \end{cases} \tag{4.2}$$

Similarly, one can define inclusion functions for **Continent**, **City**, **River**, etc.

The inclusion functions can be implemented simply using a database lookup. Large databases of geographic names are available publicly. An inclusion function can return true if a string is contained in the database of countries. An alternative implementation is to use a regular expression. As with all concrete implementations, this implementation will inevitably miss some names—particularly colloquial names. The system relies again on the law of averages to mitigate the problem.

One must take care to define the classes with knowledge of the kinds of information one is likely to find in the documents. For example, one could limit the definition of **Continent** to "North America", "South America", "Europe", "Asia", "Africa", "Australia", and "Antarctica". However, the documents one is analyzing could use a different model of the continents. Many people do not differentiate North and South America as two separate continents, and so the continent of America is disregarded. Similar troubles occur with names such as "Eurasia" or "Oceania".

In order to allow multiple names for the same entity, an optional *canonicalization* function can be defined. The canonicalization function takes a string and returns the canonical name for the entity the input string names. Example:

$$\text{CanonicalizeCountry}(str) \equiv \begin{cases} France & \text{if } str == \text{“France”}, \\ France & \text{if } str == \text{“Republic of France”}, \\ France & \text{if } str == \text{“Republique Francaise”}, \\ etc & \ldots \end{cases} \tag{4.3}$$

Similarly, one can define canonicalization functions for **Continent**, **City**, **River**, etc.

Classes are used to limit the search of a query. There are far too many possible tuples (all possible pairs of words) to perform calculations on all of them. Limiting the query to focus on the kinds of information that is important is a necessary step. It is also desirable to limit the number of tuples the system looks at.

4.1.3 Types

The system limits the focus of its search to strings that fall within a given class. Even with this limit, there are still far too many tuples to process. It becomes important to define a limit on what *tuples* the system should process.

Many of the queries that one would perform are the relations between known classes of entities. If one would like to know the capital cities of all of the countries, one is only interested in tuples that represent $\langle City, Country \rangle$ pairs. When the system is queried for the cities near bodies of water, one is interested in tuples that represent $\langle City, BodyOfWater \rangle$ pairs. Types define the range of interest.

A *type* is an ordered series of classes.

$$T \equiv \langle c_1, c_2, \ldots, c_n \rangle \tag{4.4}$$

A tuple belongs to a type if each string in the tuple belongs to the corresponding class in the type. For instance, the tuple $\langle France, Europe \rangle$ belongs to the type $\langle Country, Continent \rangle$. Because of the way classes

are defined above, the same tuple belongs to the type $\langle City, Continent \rangle$ (it is noted that "France" is also the name of many cities). One can refer to the set of tuples that belong to a type T as $\{T\}$.

Types define the kinds of entities one is interested in. They inform the system of the classes of entities that could possibly fulfill the query. However, not all possible tuples of a given type represent data that one would like to extract. Not all $\langle City, Country \rangle$ pairs represent capital cities. It becomes necessary therefore to define even more precisely what relationship one is looking for.

4.1.4 Relations

Until now, all of the definitions have been precisely defined. They all lend themselves to concrete implementations. Even ambiguity was resolved by considering equivalent sequences of characters as equal. Now, however, a way of describing human concepts must be formalized. Most text is written by a person to communicate with another human. The concepts they use are fuzzy and imprecise. It is therefore important to define a way to allow for that imprecision. This definition must allow for the widest possible variety of concepts.

As in most bootstrapping methods, the system represents those human concepts as a set of examples. The set of capital cities is represented by the set of $\langle City, Country \rangle$ pairs where the first element is the capital of the second element. The set can be complete, if all of the members of the set are known, or incomplete if only some are given.

Relations (the set of examples) are how one defines what a query should search for. They are also the result of the query. The system tries to approximate a complete set as best it can from the documents available to it.

A *relation* R is a set of tuples of a given type that define the same relationship between the elements of the tuple. A tuple can belong to more than one relation. $\langle Madrid, Spain \rangle$ belongs to the relation one might call "Capital Cities" and also to the relation "Cities in countries". $\langle Shanghai, China \rangle$ would belong to the latter and not the former.

The letter R will be used to refer to the complete set of tuples the user is querying for, whether they are known or not. The letter K is used for the subset of R that is known.

4.1.5 Sentence

Information Extraction requires unstructured text to extract information from. English and many other language break text up into sentences. The system will do the same, though it does not have to represent them as a simple series of words.

Sentences need to be represented in a way that makes it convenient to search them. The specific implementation of sentences will be defined later, since there are many possible ways it can be done. It is possible, however, to define certain properties of sentences that do not depend on the implementation.

A sentence is a representation of a natural language sentence made to be matched by a pattern. A tuple t_k is in a sentence s_i if all strings in the tuple are in the sentence.

$$t_k \in s_i \leftrightarrow \forall n(t_{kn} \in s_i) \tag{4.5}$$

Sentences are the text the system will extract information from. Of course, since there are many of them, it is useful to talk about the collection of sentences.

4.1.6 Corpus

A *corpus* C is a set of sentences.

$$C \equiv \{s_1, s_2, \cdots, s_n\} \tag{4.6}$$

Note that information about the sequence of the sentences is not maintained, nor does the system keep track of which sentences come from the same documents. All sentences are treated in the same way, though supplementing the current algorithm with that information could prove fruitful in future research.

Just as a tuple can be in a sentence, a tuple can be in a corpus.

$$t_k \in C \leftrightarrow \exists s_i (s_i \in C \wedge t_{kn} \in s_i) \tag{4.7}$$

One can express the subset of tuples in a corpus with a superscript, e.g., $\{T\}^C$ means all of the tuples of type T in the corpus C.

4.1.7 Patterns

Patterns represent lexical and syntactic features of a sentence. They are used to match tuples that have a certain grammatical relationship in a sentence. Patterns allow the system to search for syntactic features, which will later be correlated with semantic meaning.

For example, a pattern could match "**City**, **Country**", where the bold words are classes and the comma is significant. This pattern would match tuples of type $\langle City, Country \rangle$ with only a comma between them.

The patterns are generated automatically. How they are generated and represented will be addressed in Sections 5.1.2 and 5.1.3.

The system defines set of *patterns* $P \equiv \{p_1, p_2, \ldots, p_n\}$.

One expresses a matching function m that returns the set of tuples matched by pattern p_i in sentence s_j.

$$m(p_i, s_j) \subseteq \{T\}^C \tag{4.8}$$

m can return many tuples or no tuples at all (the empty set). m is constrained to not return a tuple that is not in the sentence.[1] More formally:

$$\forall p_i \forall s_j (t_k \in m(p_i, s_j) \leftrightarrow t_k \in s_j) \tag{4.9}$$

If $t_k \in m(p_i, s_j)$, then one can say that p_i matches t_k in s_j.

The system needs to know what tuples a pattern matches over the entire corpus from different sets of tuples. For instance, it might be useful to know what tuples from the relation R a particular pattern matches. The system also needs to know what tuples from the entire type T a pattern matches. A match function is defined using a set parameter G as the set over which the matches are accumulated.

If G is a set of tuples of type T, the system defines a value $match_{Gp_i}$ as the set of tuples from G matched by p_i in C. This value is the number of times p_i matches a tuple from G in the corpus. It should be noted that G is a placeholder and could be any named set.

$$match_{Gp_i} = G \cap \bigcup_{s_j \in C} m(p_i, s_j) \tag{4.10}$$

4.2 Formal Problem Definition

Now that the terms have been defined precisely, the problem can be formulated using them:

Given corpus C, T, and $K \subseteq R$, determine whether a given $t_k \in \{T\}$ is also in R, the theoretical "complete" set. A solution will give a set approximating R^C.

This is the definition of the problem this paper attempts to solve. For evaluation purposes, a relation one can have complete knowledge of (such as the capital cities of countries) will be used. However, in general, one does not have complete knowledge. In fact, if one did, there would be no need to use the system. It is useful to use the case where complete knowledge is at hand to evaluate one's algorithm. If one does not know the right answers, one cannot grade the system.

[1] In theory, a pattern can use other information besides information directly stated in the sentence (i.e., context). This is outside the scope of the current research.

4.2.1 Proposed solution

Given C, T, and $K \subset R$, generate patterns P. Create feature vectors for each tuple $t_k \in \{T\}^C$. Use those feature vectors to train a classifier. Classify the tuples in $\{T\}^C$ using the classifier. Those that are classified as in R are added to K. Repeat until no more tuples are added to K.

This algorithm is a standard bootstrapping algorithm.

Bootstrapping

Bootstrapping refers to a semi-unsupervised technique. The bootstrapping process starts with a handful of seed tuples. These seed tuples are a subset of the positive set (R). They constitute a known set of positive values.

Bootstrapping occurs iteratively. Using tuples from the seed set, the system generates patterns from the corpus. The matches of those patterns are used as features in a feature vector and trained in a classifier. Those tuples that are classified positively are added to the seed set and the process is iterated. The process stops when no new tuples are added during an iteration.

1. $K' \leftarrow K \subset R$
2. Generate patterns P using C and tuples from K'.
3. Generate feature vectors of positive matches.
4. Train classifier using K as class 1, and g randomly selected values from $\{T\}^C - K'$ as class 2.
5. Apply classifier to $\{T\}^C$.
6. Add the tuples classified as class 1 to K'.
7. if K' grew, go to step 2.
8. else return

Either the tuples in K' can be returned or the patterns in P can be returned. If the tuples are returned, the system can be considered a query system. It accepts a query definition and returns an answer in the form of a set of tuples.

If the patterns are returned from this algorithm, the bootstrapping phase can be considered a training phase. The patterns can be used later to extract more tuples from new document sources.

g can be chosen to tune the algorithm. A low value of g will tend to accept more tuples into K' with each iteration. This inevitably will accept more incorrect tuples (lower Precision). A higher value will tend to be more conservative leading to a higher Precision but a lower Recall.

When $g = 0$, *all* tuples in $\{T\}^C$ will be added to K' since no training examples from class 2 were given.

Chapter 5

Evaluation

Part of the purpose of the current research is to create a general model for the problem of Information Extraction used. It is desirable to perform many different extraction tasks with minimal human effort.

It is therefore important to show here that the algorithm achieves a high level of accuracy on a variety of extraction tasks. Consequently, the system was queried for the following relations: which countries are located in which continent and identifying the capital cities of countries.

The definition of how sentences are represented and consequently how patterns are matched is described here since they are implementation details.

5.1 Level of Parsing

Of the two most promising levels of parsing, dependency graphs were chosen over word sequences (see Section 3.1). Dependency graphs capture much more information. Also, a proper parse of a sentence can alleviate the problems of interleaving dependant clauses.

For instance, in the following sentence, the words interleaving Paris and France are unimportant to the capital city relation.

> Paris, which has some of the best restaurants in the world, is the capital of France.

In a dependency tree, the dependant clause about the quality of restaurants would be a subtree of the node for Paris. The uninteresting subtree can be ignored by the pattern. This alleviates the ordering problem, and allows a clear view of the relationship between words and phrases that are not adjacent.

5.1.1 Formal dependency tree model

A dependency tree is a tree structure with labeled nodes and labeled edges. The label at a node is a word from the sentence. The label at an edge is the grammatical relationship between the two nodes. A node may have 0 or more subtrees. The subtrees are unordered. If a node has no subtrees, it is called a leaf. All nodes but one have exactly one parent. The single node without a parent is called the root node. For more information about parsing dependency trees, see de Marneffe et al. [13].

It is necessary to define a few conventions:

One can refer to nodes with the letter n. The label of a node is $label_n$. The set of subtrees of node n $subtrees_n \rightarrow \langle edgelabel, subtree \rangle$. Subtrees of n with a specific edge label are noted $subtrees_n(label)$. The root node of s is $root_s$.

5.1.2 Pattern matching

Since it has been decided to use the dependency tree of the sentence as its representation. A pattern matching mechanism can therefore be defined.

A pattern p is a tree. Each node has a *matcher* and each edge has a label. A matcher is a predicate that takes a word and returns true to represent a match (to be defined shortly). The label at an edge is the grammatical relationship between the two nodes. Each node also has an optional return label. If the pattern matches a given tree, the word from a node in s that matches a corresponding node in p is given a label. All of these are combined into the final tuple.

For instance, a node could have the return label $arg1$. Another could have the return label $arg2$. It may be that the node with label $arg1$ matches "Paris" and the node with label $arg2$ matches "France". If the entire pattern matches, the pattern will return the tuple $\langle Paris, France \rangle$ since a tuple is constructed using the arguments in order.

If $|s|$ is the number of nodes (number of words) in s, then there are $|s|$ potential matches of a pattern p in s. Each node in s could potentially match p and contribute to the set of tuples returned by $m(p, s)$. m is extended to operate also on individual nodes ($m(p, n)$).

Intuitively, a pattern p matches a node n if the root of p matches the label of n and all of the subtrees of $root_p$ match unique subtrees of n. When matching subtrees, both the label of the node and the label of the edge have to match. Matching a pattern to a node is defined recursively and non-deterministically to make the intention more clear.

$$m(p, s) = m(p, root_s) \tag{5.1}$$

$$m(p, n) = subtrees_p = \emptyset \vee matcher_{root_p}(label_n) \wedge alltreesmatch(subtrees_p, subtrees_n) \tag{5.2}$$

where

$$alltreesmatch(j, k) = \exists_{x \in j, y \in k} m(x, y) \wedge alltreesmatch(j - x, k - y) \tag{5.3}$$

The matcher function stored at each node of the pattern tree has yet to be defined. The matcher function's role is to encode what kind of match is to be performed. Two kinds of matchers are implemented, though more are possible. The first type of matcher is the Exact match. The function matches one and only one sequence of characters. For instance, an exact match for the word "car" does not match the word "cars".

Class inclusion functions can also serve as matcher functions. This is done for the nodes in a pattern that return a value to become part of the returned tuple. This ensures that the returned tuples are of the desired type.

Other matchers can be implemented that use Wordnet hypernym data, wild-card matchers, or regular expression matchers. Integrating these matchers into the system is left to future work.

5.1.3 Pattern generation

A pattern can be generated from a tuple and a sentence with that tuple. The nodes along the shortest path between the strings in the tuple from the dependency tree are extracted. A pattern tree is created in the same form (corresponding nodes and links). Each tuple value in the shortest path corresponds to its class inclusion function in the pattern tree. All other strings correspond to exact string matchers. Figure 5.3 shows an example of creating a pattern from a sentence.

5.2 Evaluation Criteria

The standard Information Retrieval metrics detailed in Section 1.5 are used to evaluate the algorithms.

There are certain difficulties with evaluating the knowledge contained in documents that are not read and annotated by an expert. It is therefore necessary to calculate them using an objective, algorithmic metric.

Specifically, Recall (equation 1.1) is calculated as follows:

$$\text{Recall} = \frac{|K' \cap R|}{|R|} \tag{5.4}$$

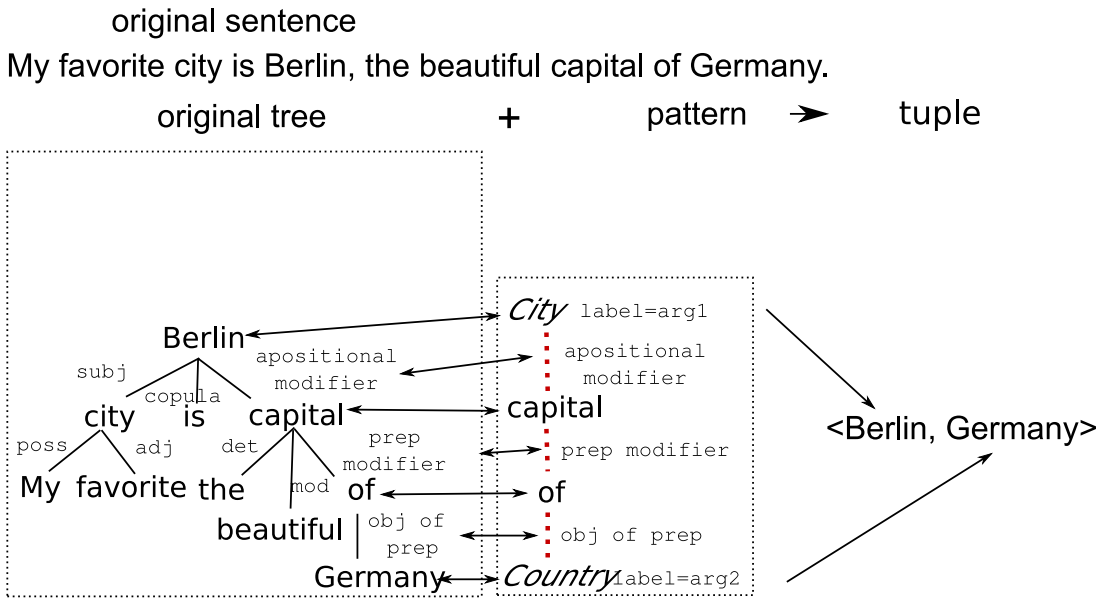

Figure 5.1: A pattern can match a sentence represented as a dependency tree. All nodes and their labels must match.

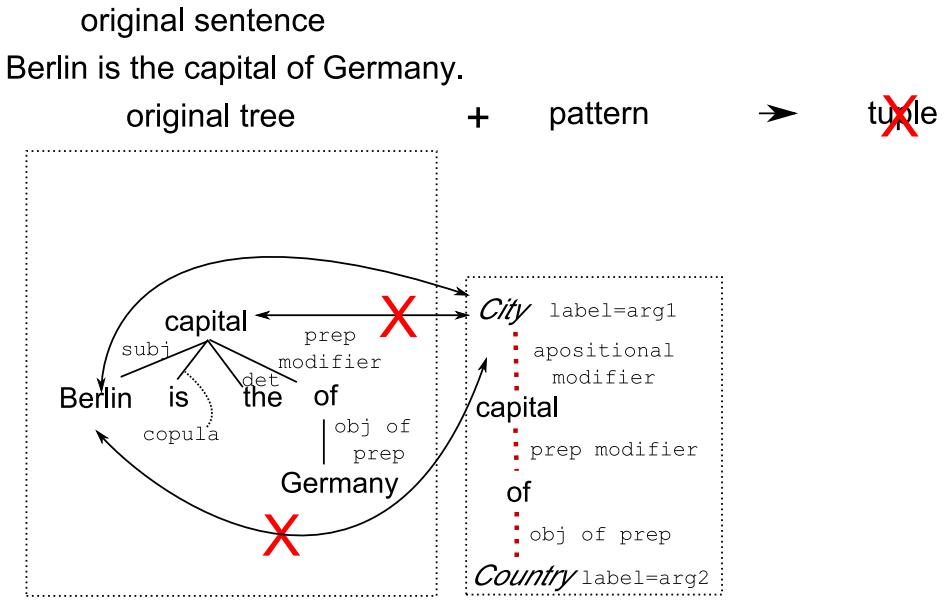

Figure 5.2: Patterns match only specific patterns. In this example, the pattern does not match the root of the dependency tree (its label is not a city) and "Berlin" does not have a subtree. The pattern fails to match.

Precision (equation 1.2) is calculated as follows:

$$\text{Precision} = \frac{|K' \cap R|}{|K'|} \tag{5.5}$$

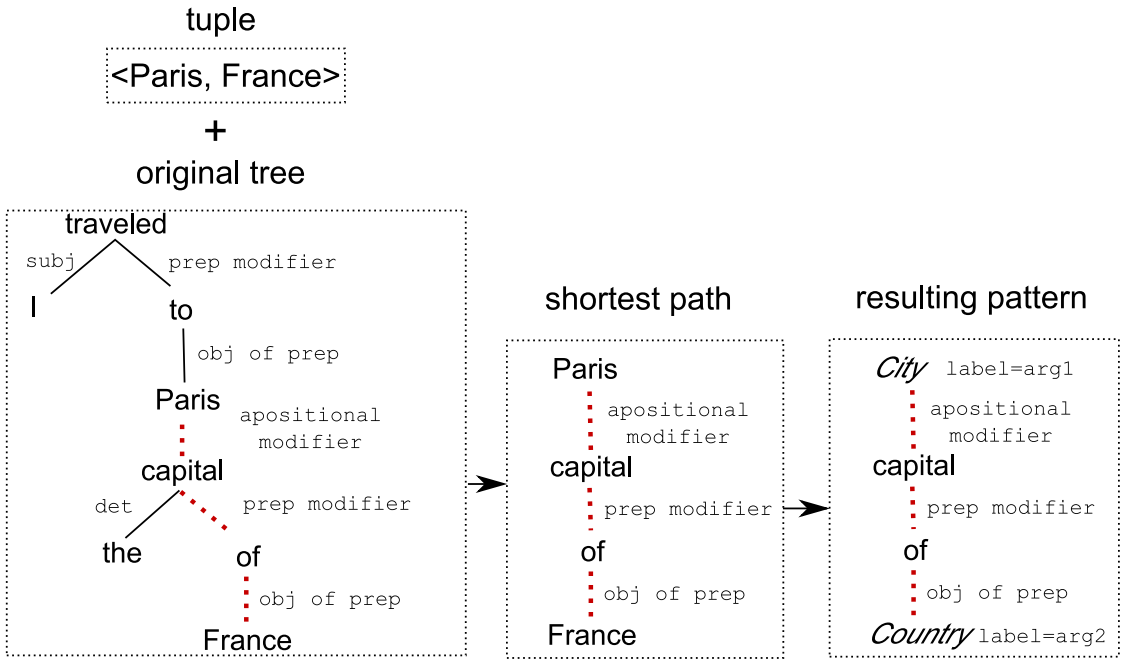

Figure 5.3: The stages of generating a pattern from a dependency tree and a tuple. Note that the values "Paris" and "France" get replaced in the pattern by their class inclusion functions.

The Recall formulation punishes the algorithm when a tuple in R is not returned by the algorithm because it is not in any sentence in the corpus. This is fair since this algorithm will need to be compared to ground truth (R)—not an unfairly selected subset of it—and a more advanced algorithm which can extract more information (i.e. an algorithm that is not limited to extracting tuples from R^C).

In order to value Precision twice as much as Recall (to value a low False Positive rate), $\beta = 0.5$ in the F-measure (equation 1.3).

5.3 Naive Bayes Classifier

The Naive Bayes Classifier is one of two classifiers used to evaluate the framework. It is important to evaluate the framework using two different classifiers. The choice of classifier is left up to the implementor of the final system. Two classifiers are tested here to evaluate whether the framework can support different classifiers and that the accuracy of the results is not due solely to the qualities of any one classifier. The specifics of the Naive Bayes Classifier will be derived here.

First, in order to use Naive Bayes, the system must assume that the patterns match statistically independent features.

Because of the above assumption, Bayes' formula holds.

$$P(t_k \in R|C) = \frac{P(C|t_k \in K')P(t_k \in K')}{P(C)} \tag{5.6}$$

and

$$P(t_k \notin R|C) = \frac{P(C|t_k \in \{T\}^C - K')P(t_k \in \{T\}^C - K')}{P(C)} \tag{5.7}$$

where C is the corpus. The category of t_k (either R or $-R$) is determined by the one that has the highest probability.

$$inR(t_k) = \begin{cases} true & P(t_k \in R|C) > P(t_k \notin R|C) \\ false & otherwise \end{cases} \tag{5.8}$$

$$P(t_k \in R|C) > P(t_k \notin R|C) \tag{5.9}$$

$$\frac{P(C|t_k \in R)P(t_k \in R)}{P(C)} > \frac{P(C|t_k \in \{T\}^C - K')P(t_k \in \{T\}^C - K')}{P(C)} \tag{5.10}$$

$$P(C|t_k \in R)P(t_k \in R) > P(C|t_k \in \{T\}^C - K')P(t_k \in \{T\}^C - K') \tag{5.11}$$

Because the patterns are independent, one can define the likelihood $P(C|t_k \in R)$ therefore as

$$\begin{aligned} P(C|t_k \in R) = P(t_k \in match_{\{T\}^C p_1}|t_k \in R) \\ \times P(t_k \in match_{\{T\}^C p_2}|t_k \in R) \\ \vdots \\ \times P(t_k \in match_{\{T\}^C p_n}|t_k \in R) \end{aligned}$$

$$P(C|t_k \in R) = \prod_{p_i \in P} P(t_k \in match_{\{T\}^C p_i}|t_k \in R) \tag{5.12}$$

And its is noted that

$$P(t_k \in match_{\{T\}^C p_i}|t_k \in R) \approx P(t_k \in match_{K' p_i}) \tag{5.13}$$

where

$$P(t_k \in match_{K' p_i}) = \frac{|match_{K' p_i}|}{|K'|} \tag{5.14}$$

Described another way, it is the proportion of tuples in K' matched by pattern p_i.

So

$$P(C|t_k \in R) = \prod_{p_i \in P} P(t_k \in match_{K' p_i}) \tag{5.15}$$

Similarly,

$$P(C|t_k \notin R) = \prod_{p_i \in P} P(t_k \in match_{(\{T\}^C - K')p_i}) \tag{5.16}$$

The initial prior is

$$P(t_k \in R) = \frac{|K'|}{|K'| + |\{T\}^C - K'|} \tag{5.17}$$

and

$$P(t_k \notin R) = \frac{|\{T\}^C - K'|}{|K'| + |\{T\}^C - K'|} \tag{5.18}$$

The inR function is now specified and will determine whether a tuple is desirable.

Practical details

There are a handful of details that must be discussed. These details arise from the practical need to implement the system in order to evaluate it.

Firstly, this is a statistical algorithm. The algorithm cannot work with a small number of sentences. It relies on the redundancy inherent in large corpora of documents.

Since all of the calculations are statistical, it is important to remove patterns that match less than a certain number of tuples. This exclusion will ensure a level of statistical certainty. For instance, a pattern that matches only one tuple will be unreliable.

Similarly, the tuples that are matched by less than a certain number of patterns should also be excluded. The statistical methods used here break down with a small number of samples. They are only accurate when the number of samples is very large. The current system throws out sentences matched by less than three patterns. And it removes patterns that match fewer than six tuples.

Secondly, there are certain minor changes to be made to the evaluation metrics. Specifically, the way they are currently calculated allows for $P(t_k \in R) = 1$ or $P(t_k \in R) = 0$. Both of these cases should be avoided, since they indicate certainty—the system is never certain. It is necessary to add in "dummy tuples" to the calculation. More specifically, four dummy tuples are added—one from each of the following cases:

1. True Positive—a tuple in R matched by the pattern
2. False Positive—a tuple not in R matched by the pattern
3. True Negative—a tuple not in R not matched by the pattern
4. False Negative—a tuple in R not matched by the pattern

Adding these dummy tuples that all patterns match will ensure that no pattern matches all seed tuples. The probability will only asymptotically approach 1 or asymptotically approach 0.

The likelihoods now become

$$P(C|t_k \in R) = \prod_{p_i \in P} \frac{|match_{R^C p_i}| + 1}{|R^C| + 2} \tag{5.19}$$

and

$$P(C|t_k \notin R) = \prod_{p_i \in P} \frac{|match_{(\{T\}^C - K')p_i}| + 1}{|\{T\}^C - K'| + 2} \tag{5.20}$$

These changes should not be considered significant to the correctness of the derivations of the formulas. These are standard statistical techniques to avoid problems that would not occur if the sample size were infinite. It should be noted that were the sample size infinite, the addition of a small constant would not change the value.

5.4 Support Vector Machines

Because the system can work for a number of different classifiers, this section will detail the use of the Support Vector Machine as a classifier.

A Support Vector Machine requires a kernel function. The kernel function chosen was the Radial Bias Function (RBF), defined as

$$RBF(x_i, x_j) = e^{-\gamma |x_i - x_j|^2} \tag{5.21}$$

This kernel function has a single parameter, γ. The Support Vector Machine itself has a second parameter, C. Using a parameter search over the data, it was found that $\gamma = 0.5$ and $C = 2.0$ were the best choices [22]. These parameters were used in the the evaluation of the algorithm.

5.5 Countries in continents

For the initial evaluation of this method, the relation of which countries are in which continent was chosen as a query. The relation is of type $\langle Country, Continent \rangle$. This needs two classes **Country** and **Continent**. The system also need to provide a corpus and specify the implementation of sentences and patterns.

5.5.1 Classes

The **Country** class contains 193 countries (each with a variety of aliases, for instance "France" and "Republic of France"—the synonyms were taken from the NGA GEOnet Names Server dataset). Any string matching one of those names is considered part of the class. Synonyms are resolved to a canonical form (using the canonicalization function) so that synonyms like "France" and "Republic of France" are equivalent.

The **Continent** class contains 6 continents—"Europe", "Asia", "Africa" "Oceania", "America", and "Antarctica". The word "Australia" is taken to mean the country to avoid the ambiguous tuple ⟨"Australia", "Australia"⟩. Alternatives names and sub-continents are canonicalized to the single word form (i.e. "East Asia" becomes "Asia"). Although East Asia is technically not a continent, a cursory examination of Wikipedia articles indicated that countries were often listed with their sub-continent and not their continent. It was considered a trivially simple improvement to the class definition.

5.5.2 Corpus

The corpus contains 8050 sentences that have a $\langle Country, Continent \rangle$ pair. These sentences were excerpted from Wikipedia.

Although in theory 1158 tuples are possible, the corpus has only 633 tuples. 193 of the tuples were from R, 441 were not from R, giving a prior of 0.30. Since there are 193 countries, R^C covers 100% of R.

5.5.3 Relation

The ground truth for the relation is defined by the CIA World Factbook. When two continents are listed (as in the case of Russia, which has a large portion in Europe but the majority in Asia), the primary location is considered correct, while the secondary continent is considered incorrect.

5.5.4 Results

The algorithm was run over the corpus with randomly selected seed sets. The seed sets were selected from the ground truth R described in section 5.5.3. Three runs of the algorithm (with different seed sets) were performed. The mean of the results of the runs was calculated. Figures 5.4, 5.5, and 5.6 show the mean results of the algorithm using the Naive Bayes Classifier with $g = 4$.

Figures 5.7, 5.8, and 5.9 show the mean results of the algorithm running with a Support Vector Machine with $g = 4$. The Support Vector Machine outperformed the Naive Bayes model. This is not surprising since the Support Vector Machine is a better classifier in general.

5.5.5 Analysis

Both clearly methods show the effectiveness of the algorithm. While maintaining high precision, the algorithm can effectively select a much large subset of R than what is initially given in the seed set.

When looking at the aforementioned figures, one might wonder why the iterative algorithm does not continue iterating indefinitely. When given three seeds, it can identify 20% of the relation. When given 40 seeds (approximately 20%) of the relation as seeds, it identifies 60% of the relation. Why, then, does it stop after 20%? The current algorithm does not stop after a certain threshold is reached, as some do. It only stops when no new tuples are added to the set.

The most plausible explanation is that the false positive tuples dampen the training of the classifier. As more tuples are added to the seed set, more unknowns are trained in the classifier. This increase in the number of training samples tends to make the classifier more accurate for the training it receives. But because the training set is becoming more and more polluted with incorrect tuples, the classifier finds a suitable class division with increasing difficulty.

The Support Vector Machine clearly outperforms the Naive Bayes Classifier. This is to be expected. However, not only does the Naive Bayes Classifier perform less accurately than the SVM, the graph does not

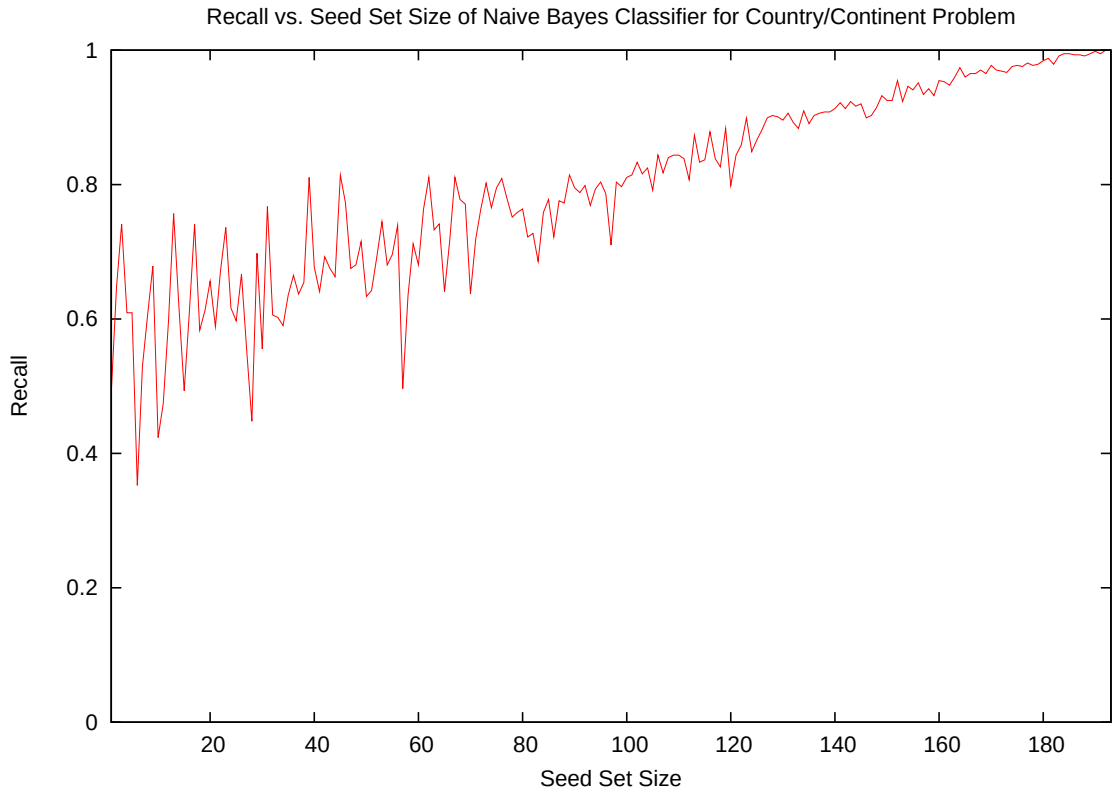

Figure 5.4: The Recall of Country/Continent relation using Naive Bayes.

present a consistent shape. This indicates that the Naive Bayes solution is erratic and should be avoided. It does not deal well with noise and propagates error.

One source of error in the results is the fact that the idea of continents is not completely clear-cut. In the case of Europe and Asia, there is an especial problem. Since Europe and Asia are not separate landmasses, there are countries that overlap. The way the data was modeled in this experiment classed countries that overlap into exactly one continent, even though they are located in two. For instance, the tuple $\langle Russia, Europe \rangle$ is considered incorrect, though a person reading it might not consider it incorrect.

Another source of error is the use of the name "Oceania" instead of the more common "Australia" in common English usage. As noted in Section 5.5.1, this was done to distinguish the name for the continent from the name of the country. Sentences that indicate countries as being in "Australia" instead of in "Oceania" were not considered as having a valid tuple in T. This could have affected the outcome.

Another source of error is in the name Eurasia to denote the combined landmass of Europe and Asia. "Eurasia" was not considered a continent name in the current model, though it certainly could have been. These problems are considered modeling problems and not problems with the Information Extraction framework presented here. The choice of model can always be made more accurate with increasing work. What is indicated, though, is that the system dealt well with the inadequacies of the models.

A final source of error is the document set itself. At base, the algorithm tries to fit the information contained in the documents to a model built from another source. There is bound to be discrepancy between the two models. This can be considered noise in the signal. The results indicate that the framework dealt well with this noise gracefully.

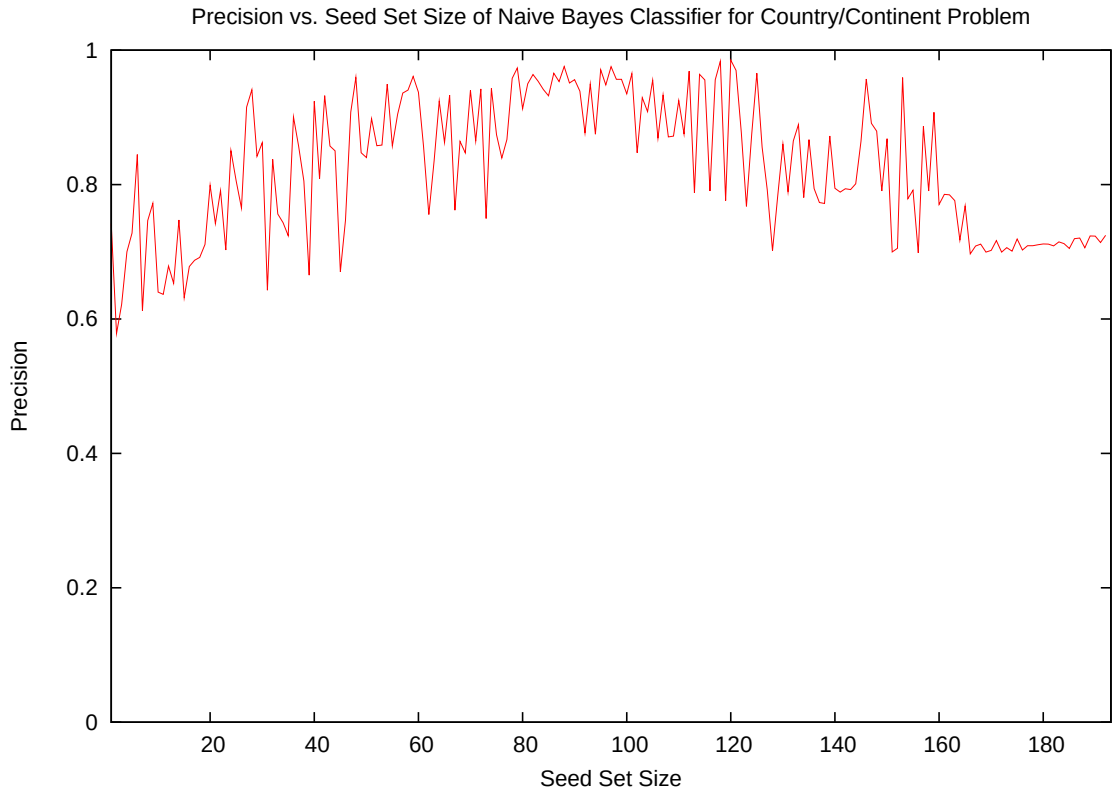

Figure 5.5: The Precision of Country/Continent relation using Naive Bayes.

5.6 Capital cities of countries

A second problem for which the algorithm was tested was a query for the capital city of each country. The relation R is of type $\langle Country, City \rangle$. Two classes need to be defined.

5.6.1 Classes

The **Country** class is defined as it was for the Country/Continent problem (see section 5.5.1).

The **City** class is defined as all names from the National Geospatial-Intelligence Agency's GEOnet Names Server database with a Feature Code of *PPLA*, *PPLC*, or *PPLG*. It contains approximately ten thousand names. Many of the names are duplicates (as in *Paris, France* and *Paris, Texas*). Many of the names are alternate spellings or alternate language forms of the same name (as in *Kiev* and *Kyiv*). No attempt to canonicalize the names is performed.

5.6.2 Corpus

The corpus contains 63652 sentences that have a $\langle Country,\ City \rangle$ pair. These sentences were excerpted from Wikipedia.

The corpus has approximately ten thousand tuples ($|\{T\}^C| \approx 10000$). All tuples from R are in the corpus. The prior for the relation is 0.018 or one in fifty-six.

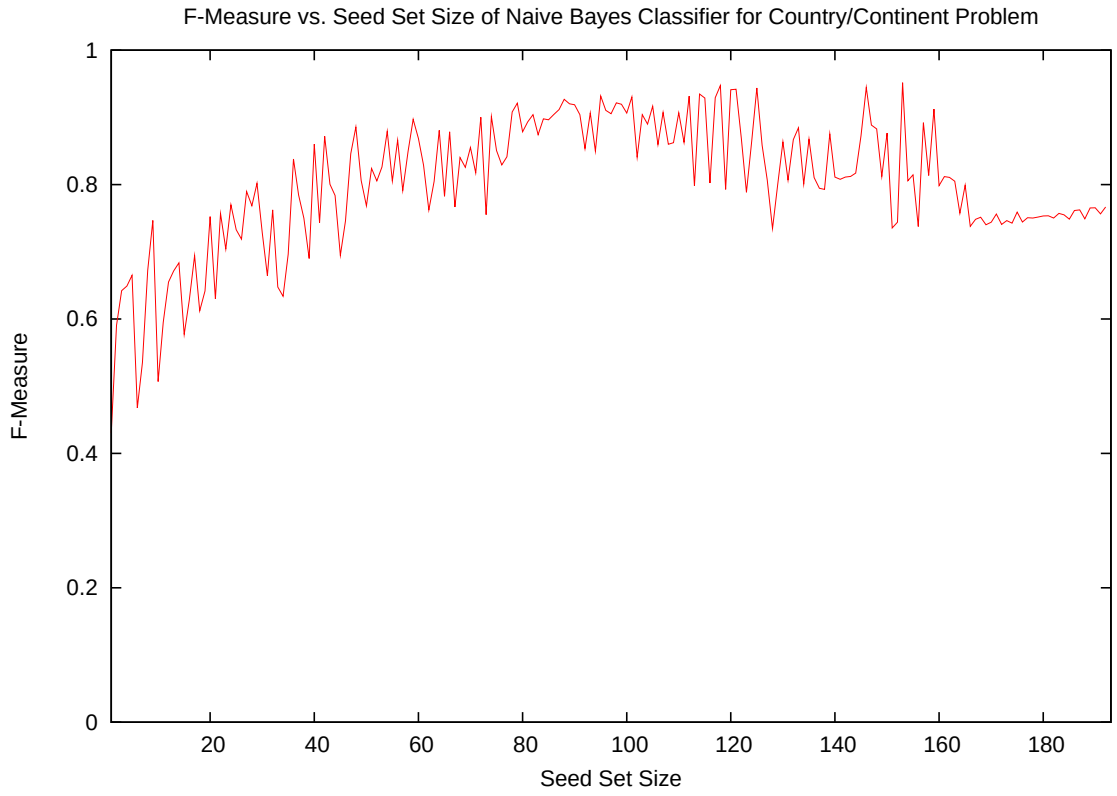

Figure 5.6: The F-Measure of Country/Continent relation using Naive Bayes.

5.6.3 Relation

The ground truth for R was taken from the CIA World Factbook. Only one capital for each country was counted. In the case where two capitals exist for the same country (Administrative and Judicial), only the administrative capital was kept. Only one possible form of the name for each capital city was chosen based on the principle spelling in the CIA World Factbook.

5.6.4 Results

The algorithm was run over the corpus with randomly selected seed sets. The seed sets were selected from the ground truth R described in section 5.6.3. Three runs of the algorithm (with different seed sets) were performed. The mean of the results of the runs was calculated. Figures 5.10, 5.11, and 5.12 show the results of the algorithm using Naive Bayes as a classifier with $g = 20$.

Figures 5.13, 5.14, 5.15 show the results of the Capital City problem using the Support Vector Machine solution with $g = 50$. Again, the Support Vector Machine has outperformed the Naive Bayes Classifier.

5.6.5 Analysis

The Naive Bayes Classifier performs significantly better than random guessing, though it shows the same lack of coherent shape as before.

The graphs from the Support Vector Machine clearly show an upward trend in the Recall with an increased seed set size. This is what one would expect from such an algorithm. As more seed values are given, the classifier has a higher confidence that the returned values are correct.

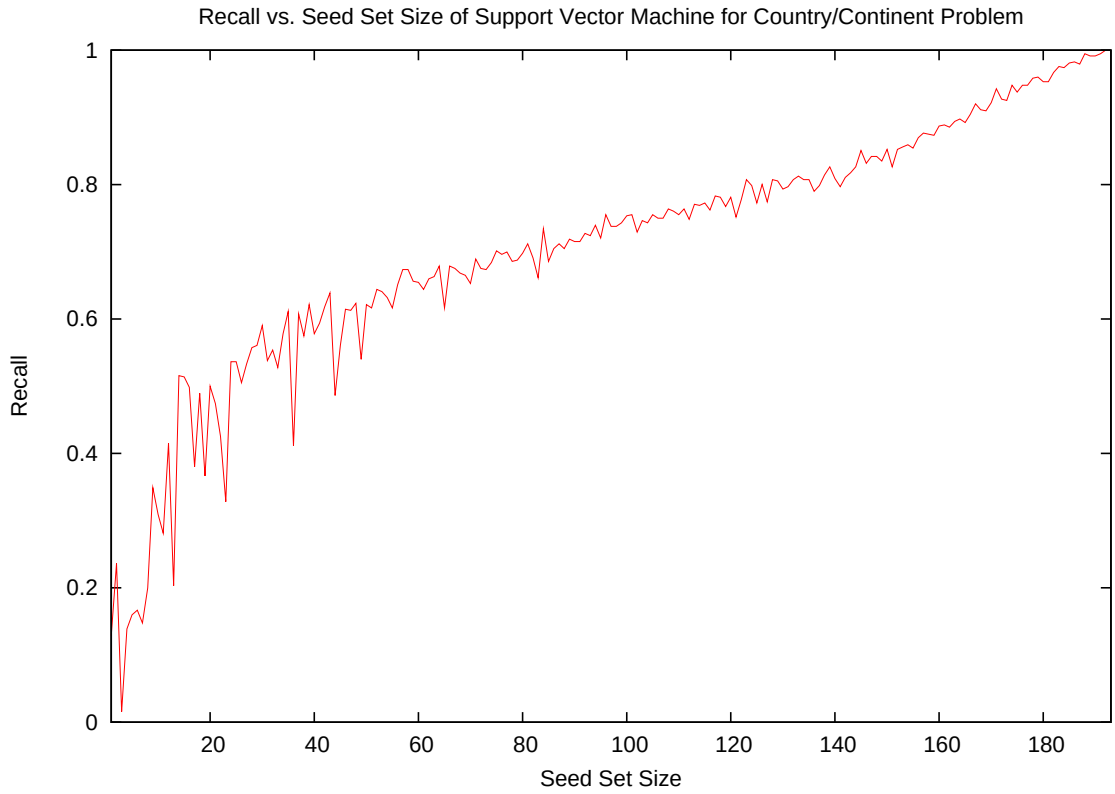

Figure 5.7: The Recall of Country/Continent relation using Support Vector Machine.

The Precision remains relatively high throughout the graph, though lower than in the previous problem (Figure 5.8). The data were analyzed and two principle reasons were found for the decrease in Precision. Firstly, the prior for this data set (1/56) is much lower than in the Country/Continent data set (1/3). This means that there is more chance for error.

The second reason is more subtle. After reviewing the returned data, it was noticed that many of what the algorithm was considering false positives were in fact alternative spellings of the correct city name. For instance, Bern, Switzerland was spelled "Berne", which is an accepted alternate spelling. Because no canonicalization was performed on the city names, and the ground truth from the CIA World Factbook spelled the city "Bern", that returned value was counted as incorrect.

The city name class was intentionally left without canonicalization. In the case of country names, canonicalization was simple and virtually error-free. Only one pair of countries had any difficulty: The Democratic Republic of the Congo and The Republic of the Congo. Both share the same alias *Congo*.

However, cities much more commonly share the same name, so it was thought unwise to perform any kind of canonicalization. Many of the false positives are incorrect. But a large number are simply alternate spellings. This small error is indicative of how the classes could be defined in the real world. However, even with this discrepancy, the results still indicate that the algorithm was able to extract correct information to a significant degree. Further, the spelling differences would not be considered errors by a human user.

The spelling discrepancies also explain the dip in Precision seen towards the end of the graph. As more examples are given to the algorithm, it finds an increasing number of matching tuples that are alternate spellings. If this duplication of data is a problem for an application, it should be avoided by canonicalizing the data or developing a more comprehensive model.

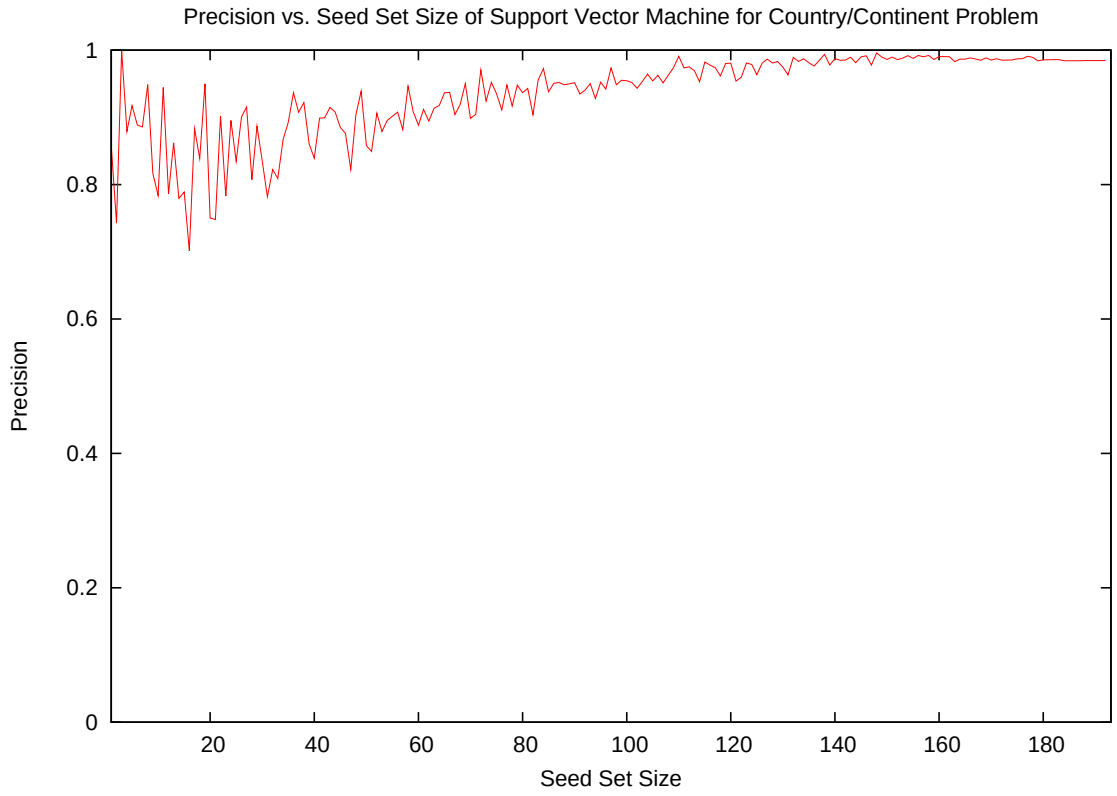

Figure 5.8: The Precision of Country/Continent relation using Support Vector Machine.

5.7 Comparisons

Comparing the results of different Information Extraction systems is difficult. Each paper presents its system run under different conditions: different corpora and different relations. A bit of analysis, therefore, is needed to compare results.

The system from prior research that is most similar to the current system is SnowBall [2]. SnowBall was evaluated at a constant seed set size of five items. The value that varied was the amount of redundancy of the five seeds in the document set. However, the results are sufficiently similar to what was found here to make a useful comparison. SnowBall's Recall varied from approximately 78% to 85%. Its Precision was approximately between 72% and 90%. The evaluation problem SnowBall used was Organization/Location. The system was queried to return where organizations were located.

Although the numeric results in this chapter do not approach the levels indicated for SnowBall, there are some mitigating factors that have to be taken into account. The Organization/Location corpus was also tested against a baseline extraction method. This method naively counted the nearest location to each organization and chooses the most frequent location as its answer. This baseline system achieved 75% Recall and over 75% Precision.

These baseline results indicate that the data was highly favored toward Organization/Location pairs that matched the relation the authors tried to extract. The SnowBall system only achieved a 15–20% increase over the naive baseline. In contrast, the current system was tested on data sets and relations whose priors were very low (1/3 and 1/56). Therefore, the discrepancy between the current results and SnowBall's results cannot be seen as a deficiency.

However, SnowBall has two main disadvantages that the current system does not have. Firstly, SnowBall requires the relation to be in a many-to-one form. It uses that constraint to develop a negative seed set.

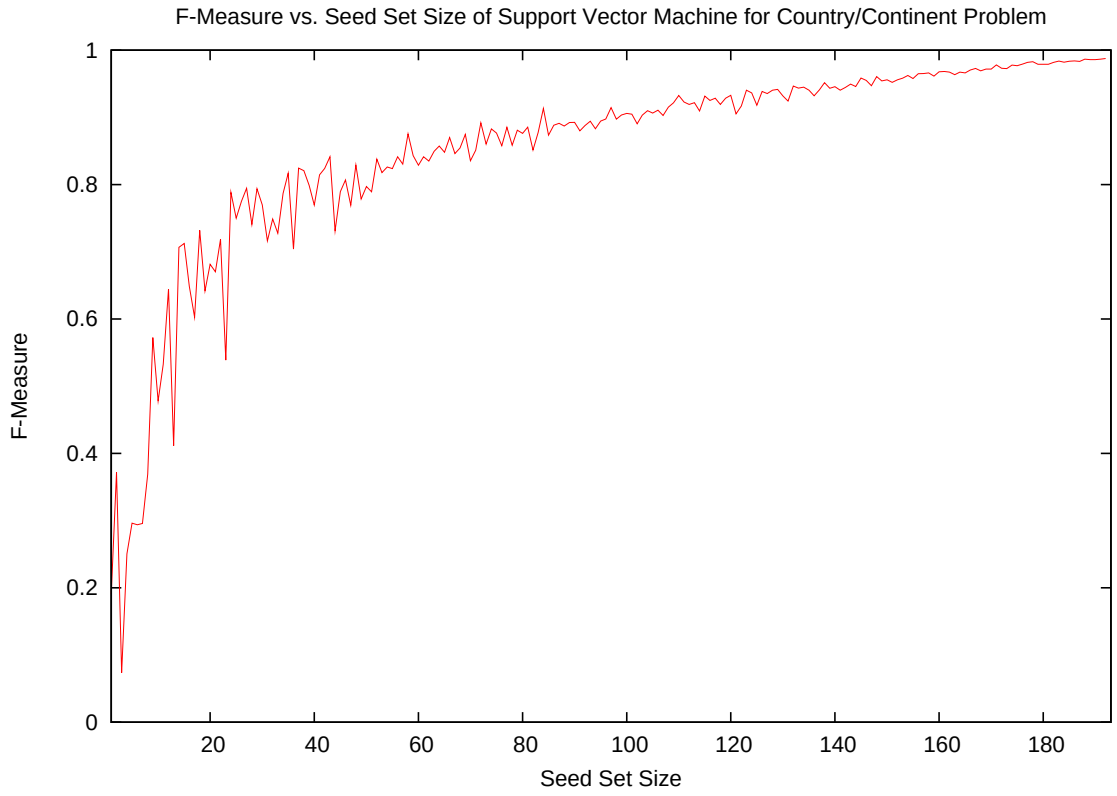

Figure 5.9: The F-Measure of Country/Continent relation using Support Vector Machine.

The current system does not have this limitation (though see section 6.3).

Secondly, SnowBall depends on a Named Entity Recognizer that has limited entity types. It only recognizes the broad categories of People, Organizations, and Locations. Although this NER system can be modified, it is expensive. The current system allows a much wider variety of entity types (defined using classes). These limitations allow the system to perform better within the limitations but cannot be as broadly applicable.

Bunescu and Mooney [4] presents a system that uses a completely supervised method. It is similar to the current system because it uses the shortest path on the dependency tree between the strings in the tuple. It would be useful to compare the current semi-supervised system to one that is completely supervised.

The system in Bunescu and Mooney [4] is evaluated on a corpus using five binary relations simultaneously. The corpus is completely annotated with the entity types and the relations. The paper assumes the entities are known (it does not discover them independently but uses those in the annotations).

The system achieved 39.2% Recall and 71.1% Precision. These results are more in-line with what was achieved in the current research. However, the current research had obstacles the system in Bunescu and Mooney [4] did not. For one, the current research identifies its own interesting entities and can be extended easily to identify new ones. Secondly, the current system does not have 100% knowledge. It achieves its level of accuracy using only a few seed values.

These comparisons, though not easily analyzed, show that the current system is an improvement over previous work.

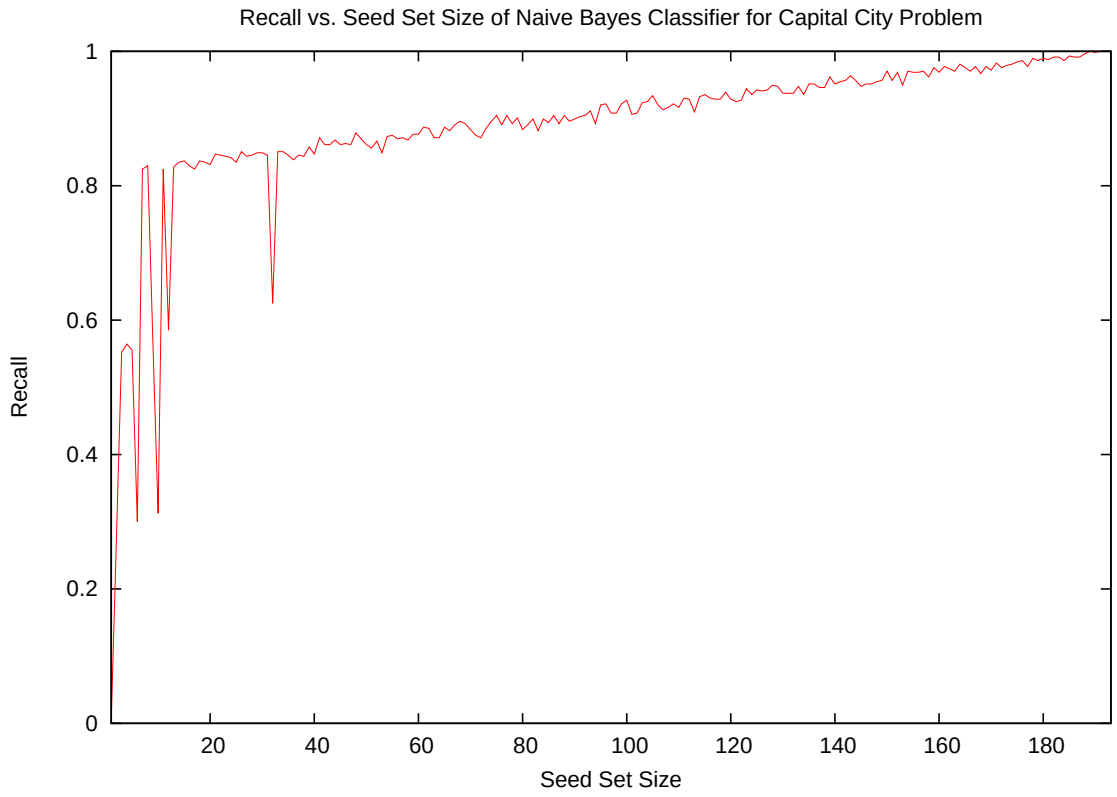

Figure 5.10: The Recall of Capital City relation using Naive Bayes

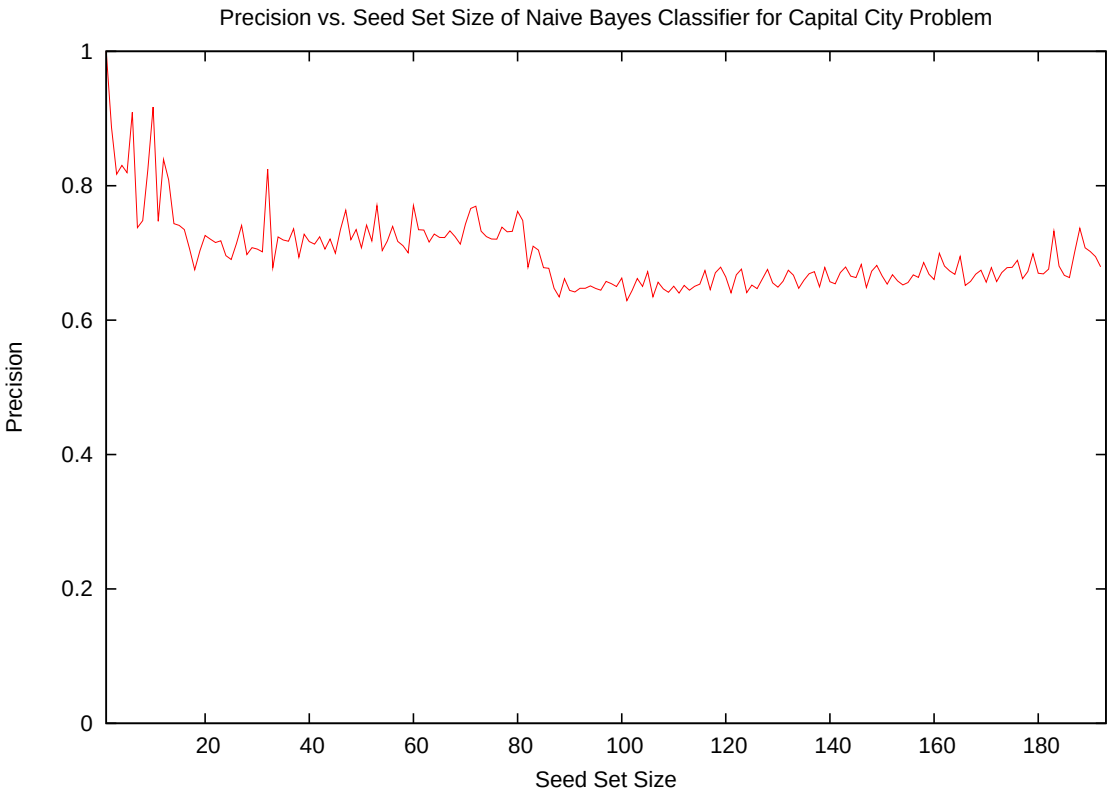

Figure 5.11: The Precision of Capital City relation using Naive Bayes

Figure 5.12: The F-Measure of Capital City relation using Naive Bayes

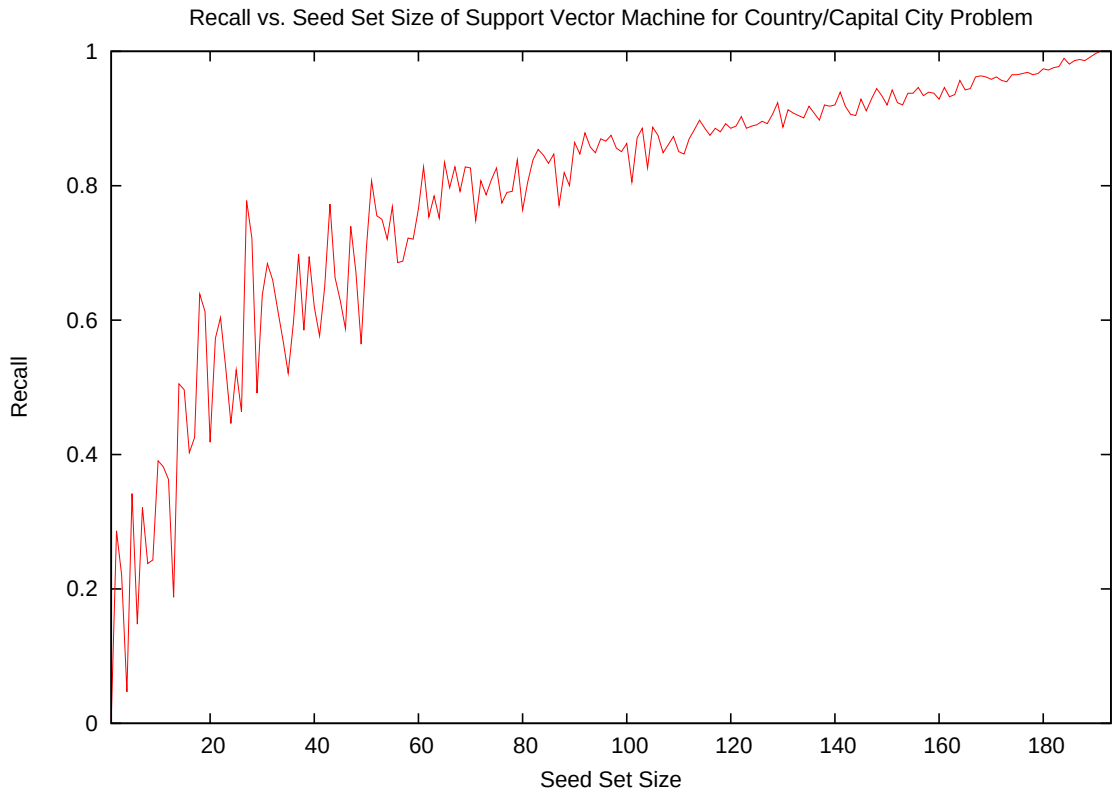

Figure 5.13: The Recall of Capital City relation using an SVM

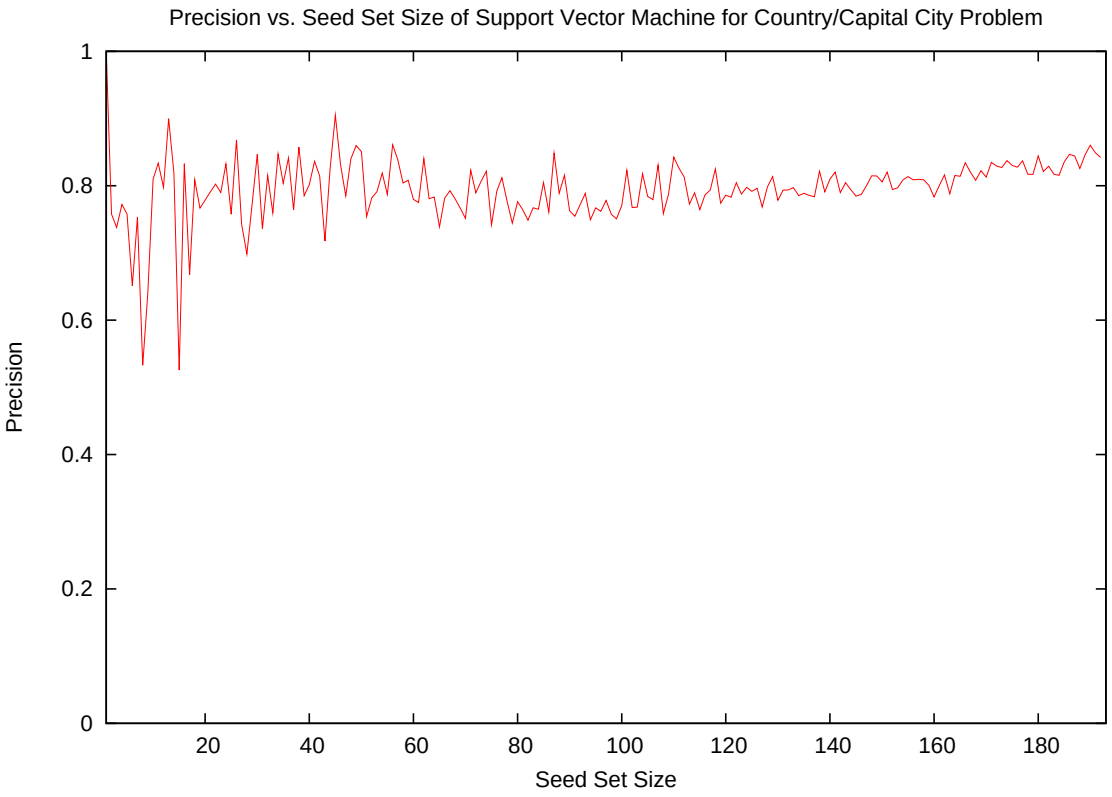

Figure 5.14: The Precision of Capital City relation using an SVM

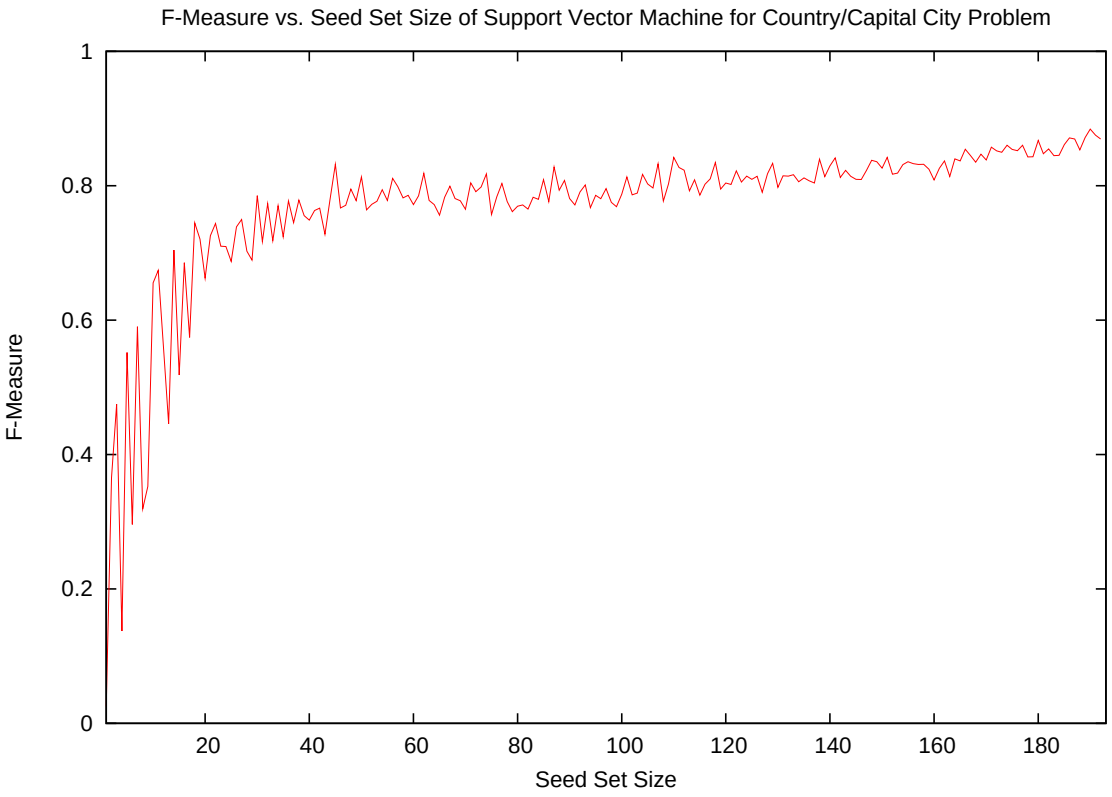

Figure 5.15: The F-Measure of Capital City relation using an SVM

Chapter 6

Conclusions

6.1 Discussion

The goal of this paper was to develop an Information Extraction framework that could extend easily to many domains of differing degree. This was further constrained by a need to respond to user-generated queries for specific information—as opposed to unconstrained clustering of results. The system needs to give a definitive answer whether a tuple is relevant.

The system developed here succeeded in those goals and has several advantages above and beyond the original objectives. The framework is not dependant on the type of classifier nor on the level of parsing. This lets the implementor decide what kind of parsing is best for the particular application and what kind of classifier would best suit the needs. In cases where performance is a concern, lower-cost algorithms can be used.

One of the stated goals was to develop a system that was not constrained to the arity of the relation it supported. Although the framework was not tested with tuples other than binary, the system is trivially extended to those cases.

Another interesting property is that the system can be tuned using the g parameter (see Section 4.2.1). A lower g value will make the system more zealous with its suggestions—at the risk of Precision. A higher g value will make the system more conservative—causing a lower Recall.

The results showed that the system can deal with noise in the query (poor or incomplete modeling of the problem) and noise in the document set (Wikipedia contains factual errors). This kind of noise is to be expected from the type of text typically found on the Internet.

6.2 Limitations

There are several potential limitations to the definition of the framework presented here. The KnowItAll system [15] can discover new members of a class by using open-ended patterns. For instance, to find new names of plants, it searches its corpus for the phrase "plants like . . ." and considers the resulting strings.

The current system assumes that the classes of strings that can be answers to a query are known. They do not have to be completely enumerated—the class inclusion function can be defined as a regular expression or other text matching function. However, using a wild-card inclusion function (which constantly returns true) would likely violate the assumption that a highly structured query will eliminate ambiguity (see Section 3.2.3). Using a wild-card inclusion function needs to be tested.

Another feature that needs to be tested is *reflexive relations*. In the Country/Continent problem, "Australia" was defined as a country and not a continent in order to avoid the tuple $\langle$"*Australia*", "*Australia*"$\rangle$. A relation between two entities of the same type could in theory be extracted. For example, a relation of type $\langle$*Country, Country*$\rangle$ is reflexive. This kind of relation could show problems. One obvious problem is

that any sentence that has a *Country* will have a tuple of type ⟨*Country, Country*⟩. This might affect the efficiency of the algorithm with a large number of useless tuples. Further research is required.

A limitation in the testing presented in Chapter 5 is that no relations other than binary are tested. It is assumed that ternary and higher-order relations are trivial extensions, but this was not tested.

6.3 Future Work

There are several interesting extensions to the system that could prove promising.

The first is to test the system using no parsing. A pattern matching scheme could be developed as in Agichtein and Gravano [1], Bunescu and Mooney [5], Riloff [33], and Jie and Min [24] that works merely on the words in the interleaving text of the tuple. It could be the case that the extra effort of parsing is not justified given its high cost. The most promising of those methods is Bunescu and Mooney [5], which counts the number of subsequences that match between two sequences of words.

Agichtein and Gravano [1] and Suchanek et al. [37] use a negative seed set as well as a positive seed set. The current work could be extended to use knowledge of what is not desired to help find what is desired. The classification can be trained to identify three classes: Positive, Negative, and Unknown. This could prove fruitful.

If the previous negative-seed-set extension is implemented, another powerful extension is possible. SnowBall [1] uses an implied negative seed set created from the positive seed set. Since it only supports many-to-one or one-to-one relationships, SnowBall can infer values that cannot be correct. For instance, if it knows that Paris is the capital of France, it can infer that Madrid is not the capital of France.

It is contrary to the goals of the system to force such a restriction on the queries. However, in the case where the user knows the type of relation (one-to-one, many-to-one, or many-to-many), the query can include this as an optional parameter. An inferred seed set can be created automatically and used in the extension described previously.

Another possible extension is to build the system to not require that the tuple be in a sentence. Some simple model of context could be created, where the subject of the document could always be implied and matched by a special part of the pattern. Alternatively, the context can contain the last values of each class mentioned in the text. When a pattern needs to match a *Country*, for instance, and it matches the word "it", the pattern would use the *Country* stored in the context. This might increase the accuracy of the framework. It could also merely complicate it with out much gained.

The system can also be made to use an active learning approach as in Culotta and McCallum [9] and Soderland [35]. Those systems queried the user for more training examples. The current system has an advantage, though, in that it extracts new tuples with every bootstrapping iteration. The extraction step can be performed in a separate process from the user interface. The values it extracts can be fed to the user interface as they are generated. The user can then select which values he/she knows to be correct or incorrect, and add them to the appropriate seed set. As the user's seed set changes, the extraction system can be restarted with those values. This would effectively build the initial seed set interactively with the user.

As the number of patterns increases, the size of the feature vector increases, as well. A large number of features makes training the classifier more costly, even if the features contribute very little to the accuracy of the classifier. A branch of data mining called *feature subset selection* could play a role in finding those features that are most important to the classification [38]. A feature subset selection stage could be added after the bootstrapping process is complete.

Finally, work needs to be done into how the g parameter (see Section 4.2.1) should best be chosen. Currently, it is chosen interactively until the best Recall/Precision ratio is found. It might be automated to relieve the burden on the user. Clearly, there is much more room for exploration using this framework.

Bibliography

[1] E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collections. *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, 2000.

[2] E. Agichtein, E. Eskin, and L. Gravano. Combining Strategies for Extracting Relations from Text Collections. *Proceedings of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2000), May*, 2000.

[3] S. Brin. Extracting patterns and relations from the world wide web. *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT98*, pages 172–183, 1998.

[4] R.C. Bunescu and R.J. Mooney. A shortest path dependency kernel for relation extraction. *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 724–731, 2005.

[5] R.C. Bunescu and R.J. Mooney. Subsequence kernels for relation extraction. *Advances in Neural Information Processing Systems*, 18:171–178, 2006.

[6] M.E. Califf and R.J. Mooney. Relational learning of pattern-match rules for information extraction. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 328–334, 1999.

[7] M.E. Califf, R.J. Mooney, and D. Cohn. Bottom-Up Relational Learning of Pattern Matching Rules for Information Extraction. *Journal of Machine Learning Research*, 4(2):177–210, 2004.

[8] A. Culotta, A. McCallum, and J. Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 296–303, 2006.

[9] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 746–751. AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X.

[10] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[11] Nikolai Daraselia, Anton Yuryev, Sergei Egorov, Svetalana Novichkova, Alexander Nikitin, and Ilya Mazo. Extracting human protein interactions from medline using a full-sentence parser. *Bioinformatics*, 20(5):604–611, 2004. ISSN 1367-4803.

[12] E. de Argaez. Internet world stats. *Internet world stats*, 15(3).

[13] M.C. de Marneffe, B. MacCartney, and C.D. Manning. Generating typed dependency parses from phrase structure parses. *LREC 2006*, 2006.

[14] E. Dimitriadou, A. Weingessel, and K. Hornik. A mixed ensemble approach for the semi-supervised problem. *Proceedings of the International Conference on Artificial Neural Networks*, pages 571–576, 2002.

[15] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.M. Popescu, T. Shaked, S. Soderland, D.S. Weld, and A. Yates. Web-scale information extraction in knowitall:(preliminary results). *Proceedings of the 13th international conference on World Wide Web*, pages 100–110, 2004.

[16] O. Etzioni, M. Cafarella, D. Downey, A.M. Popescu, T. Shaked, S. Soderland, D.S. Weld, and A. Yates. Methods for Domain-Independent Information Extraction from the Web: An Experimental Comparison. *Proceedings of the AAAI Conference*, pages 391–398, 2004.

[17] K. Fundel, R. Kuffner, and R. Zimmer. RelEx–Relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365, 2007.

[18] R. Grishman. Information Extraction: Techniques and Challenges. *Information Extraction (International Summer School SCIE-97)*, 1997.

[19] R. Grishman and B. Sundheim. Message Understanding Conference-6: a brief history. *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 466–471, 1996.

[20] S. Harabagiu, C.A. Bejan, and P. Morarescu. Shallow semantics for relation extraction. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1061–1066, 2005.

[21] T. Hasegawa, S. Sekine, and R. Grishman. Discovering Relations among Named Entities from Large Corpora. *Proceedings of the Annual Meeting of Association of Computational Linguistics (ACL)*, 2004.

[22] C.W. Hsu, C.C. Chang, C.J. Lin, et al. A practical guide to support vector classification. *National Taiwan University, Tech. Rep., July*, 2003.

[23] Scott B. Huffman. Learning information extraction patterns from examples. In *Learning for Natural Language Processing*, pages 246–260, 1995.

[24] Z.G.D.S.U.J.Z. Jie and Z. Min. Exploring Various Knowledge in Relation Extraction. *Ann Arbor*, 100, 2005.

[25] J.D. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning table of contents*, pages 282–289, 2001.

[26] Y. Liu, Z. Shi, and A. Sarkar. Exploiting Rich Syntactic Information for Relation Extraction from Biomedical Articles. *Procs. of NAACL/HLT*, 2007.

[27] P. Lyman, H.R. Varian, et al. How much information. *at http://www. sims. berkeley. edu/how-much-info*, 2003.

[28] G.S. Mann and D. Yarowsky. Multi-field information extraction and cross-document fusion. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 483–490, 2005.

[29] M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[30] M.F. Moens. *Information extraction: algorithms and prospects in a retrieval context*. Springer, Dordrecht, 2006.

[31] P. Pantel and M. Pennacchiotti. Espresso: leveraging generic patterns for automatically harvesting semantic relations. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 113–120, 2006.

[32] M. Pennacchiotti and P. Pantel. A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. *Proceedings of Inference in Computational Semantics (ICoS-06), Buxton, England*, 2006.

[33] E. Riloff. Automatically Generating Extraction Patterns from Untagged Text. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 2:1044–1049, 1996.

[34] Y. Shinyama and S. Sekine. Preemptive information extraction using unrestricted relation discovery. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 304–311, 2006.

[35] S. Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1):233–272, 1999.

[36] M. Stevenson and M.A. Greenwood. A Semantic Approach to IE Pattern Induction. *Ann Arbor*, 100, 2005.

[37] F.M. Suchanek, G. Ifrim, and G. Weikum. LEILA: Learning to Extract Information by Linguistic Analysis. *Proceedings of the ACL-06 Workshop on Ontology Learning and Population*, 2006.

[38] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2005.

[39] P.D. Turney. Expressing Implicit Semantic Relations without Supervision. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 313–320, 2006.

[40] CJ Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann Newton, MA, USA, 1979.

[41] J. Xiao, J. Su, G. Zhou, and C. Tan. Protein-Protein Interaction Extraction: A Supervised Learning Approach. *Proc Symp on Semantic Mining in Biomedicine*, pages 51–59, 2005.

Vita

Eric Normand was born in New Orleans, Louisiana, to Rick Normand and Elizabeth Williams. After attending Benjamin Franklin High School, he went on to study the connections between Computer Science, Philosophy, and Linguistics at the University of New Orleans, where he earned his Bachelor of General Studies. Before completing his Master's degree requirements, Mr. Normand served in the Peace Corps for two years during which he taught secondary school Mathematics in a remote village in Guinea, West Africa.