A Semi-Supervised Approach to the Construction of Semantic Lexicons

by

Mohamad Hasan Ahmadi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A growing number of applications require dictionaries of words belonging to semantic classes present in specialized domains. Manually constructed knowledge bases often do not provide sufficient coverage of specialized vocabulary and require substantial effort to build and keep up-to-date. In this thesis, we propose a semi-supervised approach to the construction of domain-specific semantic lexicons based on the distributional similarity hypothesis. Our method starts with a small set of seed words representing the target class and an unannotated text corpus. It locates instances of seed words in the text and generates lexical patterns from their contexts; these patterns in turn extract more words/phrases that belong to the semantic category in an iterative manner. This bootstrapping process can be continued until the output lexicon reaches the desired size.

We explore employing techniques such as learning lexicons for multiple semantic classes at the same time and using feedback from competing lexicons to increase the learning precision. Evaluated for extraction of dish names and subjective adjectives from a corpus of restaurant reviews, our approach demonstrates great flexibility in learning various word classes, and also performance improvements over state of the art bootstrapping and distributional similarity techniques for the extraction of semantically similar words. Its shallow lexical patterns also prove to perform superior to syntactic patterns in capturing the semantic class of words.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Olga Vechtomova, for the generous support, guidance and dedication she has provided in this endeavor.

I am also heartily grateful to Charlie Clarke and Gordon Cormack for accepting to read this thesis.

I would like to thank Ashif Harji and Maheedhar Kolla, graduate students in the Programming Languages and Information Retrieval groups, for always being available to help; and also Kaheer Suleman for helping with parts of the implementations.

Last but not least, my most sincere thanks go to my beloved family for their unconditional love and support and for encouraging me to reach further.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many Natural Language Processing (NLP) tasks such as Information Extraction (IE), Question Answering (QA), etc. benefit from semantic resources. One of such resources is a semantic lexicon or simply a dictionary of words (word senses) where each entry is labeled with one or more semantic classes (hypernyms), e.g., *pizza* is a FOOD and *ford* is a CAR MAUFACTURER. The purpose of Information Extraction (IE) systems is to extract domain-specific information from natural language text. Corpus-based approaches to IE typically rely on a domain-specific semantic lexicon. For instance, a system that extracts customers' opinions about features of digital cameras from review texts can benefit from a lexicon of features offered in digital cameras.

Domain-specific semantic lexicons may be constructed by hand or automatically. To manually build and maintain domain-specific lexicons is both time-consuming and costly, since it has to be performed by domain experts. This effort seems even less cost effective when one considers the fact that such a lexicon is useful for only one type of IE task and for only one domain.

Alternatively, a number of broad-coverage lexical dictionaries and semantic lexicons such as WordNet (Miller, et al. 1990) and Cyc (Lenat, Prakash and Shepherd 1985) are available in the public domain. However, despite constant growth of some resources such as WordNet, they are often short of specific vocabulary and jargon used in specialized domains (e.g., Medicine or Technology) as well as abbreviations and spelling alternatives. Roark and Charniak (Roark and Charniak 1998) report 60 percent of the words extracted by their semantic lexicon constructor were not present in WordNet. For instance, in our corpus of restaurant reviews "Dog" is widely used to refer to "Hot dog" or "Sunday" is a common misspelling of "Sundae" but neither of these senses are covered in WordNet. This makes broad-coverage semantic lexicons unfit for IE in specialized domains.

These observations suggest developing methods for automated construction of semantic lexicons for specialized domains. Here, *semantic lexicon construction* means to create a list of words that belong to a specific semantic class. This task is also referred to as *lexical acquisition*, *hyponym learning*, and *semantic class induction*. These methods can be used not only to produce semantic lexicons for specialized domains, but also to improve existing resources such as WordNet with new words and senses.

In this thesis, we propose a semi-supervised bootstrapping approach to automatic semantic lexicon construction. Our method extracts members of the target semantic category based on collective information from a large collection of extraction pattern contexts. The inputs to our method are a small set of seed words representing the target semantic category and an unannotated text corpus. It locates instances of the seed words in the corpus and produces extraction patterns from their contexts; these patterns extract more words/phrases that belong to the semantic category and the bootstrapping process continues with larger set of category members.

As opposed to state of the art approaches to this task, our method does not incorporate syntactic information or heuristics in extraction patterns and simply uses windows of words around instances of category members for this purpose. This use of purely lexical extraction patterns has a number of advantages. For instance:

1. They are inexpensive to obtain as the corpus does not need to be parsed. This makes our method scalable to large corpora.
2. Gives us the flexibility to learn semantic lexicons of words belonging to various parts-of-speech (POS).
3. Allows capturing occurrences of category members in high-confidence contexts such as lists, which may not be possible using syntactic relations among words.

A by-product of our method can be a set of extraction patterns which are highly associated with the target semantic category. This set of domain-specific lexical patterns is another resource which (in addition to a semantic lexicon) comes in handy for IE systems; they can extract category members when applied to any *unannotated* corpus in the same domain or even Web documents.

Instead of simply extracting nouns or noun phrases (NPs) as output by phrase chunkers, we propose methods for detecting multi-word terms of the target category nested within NPs. Our method features learning multiple semantic classes simultaneously as well as other techniques such as using feedback from competing classes to learn patterns which are more highly associated with the semantic categories.

And finally, we propose an efficient implementation which allows handling a large number of extraction patterns.

We evaluate our method for extraction of dish names and subjective modifiers from a corpus of 157,865 restaurant reviews. We evaluate different components of our method aiming to constrain and guide the bootstrapping process, and compare their performance against those of a state of the art bootstrapping method (Thelen and Riloff 2002). We also evaluate our bootstrapping method against a different approach to semantic lexicon construction (Vechtomova and Robertson 2011), where contexts are not used as extraction patterns but as similarity features used to rank words according to their similarity to a set of seeds.

The main contributions of this work can be summarized as follows:

1. Introducing a powerful and low-cost semi-supervised method and an efficient implementation for construction of domain-specific semantic lexicons.
2. Proposing methods to control *category drift*[1] in bootstrapping a semantic lexicon.
3. Construction of a corpus of restaurant reviews and accompanying evaluation set, labeled with dish names, aspects of restaurants, and subjective modifiers/expressions suitable for IE tasks such as Semantic Lexicon Induction or Sentiment Analysis.
4. Proposing methods to identify valid multi-word category members. This is most important in opinionated data where NPs, as output by NP-chunkers, often are not suitable for addition to the lexicon.
5. Comparing lexical patterns (window of words), as a cheaper alternative, against syntactic patterns (based on syntactic roles of words) in terms of their power to capture information about semantic classes of words.
6. Comparing the effectiveness of using words' contexts as extraction patterns to locate occurrences of category members (bootstrapping methods) against using them as features to infer semantic similarity between words (context vector models).

[1] We also refer to it as "drift" for brevity.

This thesis is organized as follows. Chapter 2 discusses previous work on automated methods for semantic lexicon induction. Chapter 3 overviews our approach: we present our bootstrapping algorithm for corpus-based semantic lexicon induction and explain how it is different from a state of the art bootstrapping method. We also describe various components added to the method for increasing the learning precision. Chapter 4 starts with the process of preparing our text corpus of restaurant reviews and construction of an evaluation set. It presents implementation details and techniques used to make the method more scalable and efficient. It also outlines the experiments we performed and their goals. Chapter 5 presents the empirical results showing that our method outperforms other techniques employed in previous works to learn semantic lexicons. It also demonstrates the effectiveness of our lexical patterns compared to syntactic patterns as well as their flexibility in extraction of various word classes: dish names (noun phrases) and subjective modifiers (adjectives). Chapter 6 concludes the work and contributions and discusses possibilities for future work.

Chapter 2

Background and Literature Review

An extensive amount of work has been done on automated semantic lexicon construction. These methods can be divided into two groups: corpus-based methods and Web-based methods. Corpus-based methods are generally designed to learn domain-specific semantic lexicons from a collection of domain-specific texts. On the contrary, despite the fact that the Web is a huge repository of knowledge containing specialized terminology for presumably any domain, Web-based methods are typically designed to construct broad-coverage semantic lexicons (similar to WordNet).

However, there has been efforts to combine the best of the two worlds by either incorporating more accurate Web statistics into corpus-based semantic lexicon induction methods (Igo and Riloff 2009), or trying to find the right corner of the Web pertaining to a specialized domain (Vechtomova and Robertson 2011).

A broader classification of different approaches to automatic semantic lexicon construction may be based on the amount of training data they require, which groups them into three categories of **unsupervised**, **semi-supervised**, and **supervised** methods. Although automatic approaches offer significant efficiency compared to manual construction of semantic lexicons, the cost of annotating a training corpus is nontrivial. Therefore, semi-supervised and unsupervised approaches have received more attention in this trend of research:

2.1 Semi-supervised and Unsupervised Approaches

There has been some work on fully unsupervised semantic clustering of words ((D. Lin 98), (Lin and Pantel 2002), and (Davidov and Rappoport 2006)). Clustering methods though, may not always produce desired granularities of semantic classes and do not associate a semantic label to the clusters. Related

works in fully automated ontology construction, however, create hierarchies of semantic categories (e.g., (Caraballo 1999) and (Cimiano and Volker 2005)).

On the other hand, semi-supervised methods use a small amount of labeled data (a handful of category members known as "Seeds" or "Seed words") which is inexpensively obtainable, but results in considerable improvements in learning accuracy. Furthermore, this enables the user to identify the desired granularity of the semantic class by providing the appropriate seeds.

Both semi-supervised and unsupervised approaches try to group a set of words which belong to a semantic category by using some measure of *semantic similarity* between them. Different methods have been proposed to capture this notion each having their strengths and weaknesses. Here we plan to cover the major trends and provide the necessary background in each subsection:

2.1.1 Statistical Co-occurrence Measures

Several statistical tests have been proposed attempting to measure association or similarity between words by using statistics taken from specialized corpora or the Web. Drawing statistics from specialized corpora, however, usually suffers from lack of accuracy due to their relatively small sizes. Regardless of the source, a key assumption in co-occurrence statistics is that semantic similarity between words is a result of co-occurrence (closeness of the words in text). (Terra and Clarke 2003) looked into variety of ways to estimate word co-occurrence frequencies in text and investigated their impact on the performance of word similarity measures. Be it text window or document, the notion of co-occurrence of two words can be represented by a contingency table as shown in Table 1.

	w_1	$\neg w_1$	
w_2	f_{w_1,w_2}	$f_{\neg w_1,w_2}$	f_{w_2}
$\neg w_2$	$f_{w_1,\neg w_2}$	$f_{\neg w_1,\neg w_2}$	$f_{\neg w_2}$
	f_{w_1}	$f_{\neg w_1}$	

Table 1: Contingency Table

w_i and $\neg w_i$, respectively, indicate presence or absence of word in a given text window or document. Each cell in the table represents the joint frequency $f_{w_i,w_j} = N_{max} \times P(i,j)$, where N_{max} is the maximum

number of co-occurrences. Assuming words occur independently of each other, the probability values of the cells are calculated as shown in Table 2.

$P(w_1, w_2) = P(w_1) \times P(w_2)$
$P(w_1, \neg w_2) = P(w_1) \times P(\neg w_2)$
$P(\neg w_1, w_2) = P(\neg w_1) \times P(w_2)$
$P(\neg w_1, \neg w_2) = P(\neg w_1) \times P(\neg w_2)$

Table 2: Probabilities assuming independence

Here we briefly introduce some of the most commonly used word similarity measures and their application in the field of semantic lexicon construction. These methods exploit different measures of how distant observed frequency values of the contingency table are from expected values under an independence assumption (Terra and Clarke 2003). According to (Tan, Kumar and Srivastava 2002) the differences between the methods arise from non-uniform marginals and how they deal with this non-uniformity.

2.1.1.1 Pointwise Mutual Information (PMI):

PMI, introduced by (Church, Gale and Hanks 1991), is an Information-Theoretic measure of word similarity based on word co-occurrence.

$$PMI(W_1 = w_1, W_2 = w_2) = log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)} \quad \text{Eq. 1}$$

Simply stated, PMI is a rough measure of how much one word tells us about the other. A positive value indicates w_i and w_j occur together more than expected under independence assumption, zero indicates independence, and a negative value indicates each word tends to appear only when the other does not.

(Riloff and Shepherd 1997) take a semi-supervised statistical approach for building semantic lexicons, based on the observation that category members tend to be surrounded by other category members in text. Examples are conjunctions (lions and tigers and bears), lists (lions, tigers, bears...), appositives (the stallion, a white Arabian), and nominal compounds (Arabian stallion; tuna fish).

The input to their algorithm is a set of five known category members and a noun phrase chunked text corpus. They collect windows of one noun phrase to the left and one of noun phrase to the right of all noun phrases having a known category member as their head. Every word *W* appearing in these context

windows receives a score for membership in that category based on a conditional probability that the word appears in the contexts of category members:

$$Score(W, C) = \frac{freq(W\ in\ C's\ context\ windows)}{freq(W)} \quad \text{Eq. 2}$$

Note that the PMI score, when used for ranking, often is replaced by the above equation for ease of computation by replacing the probabilities with frequencies and removing *log* and $freq(C'scontext\ windows)$ in the denominator simply because they do not affect ranking.

Words with a corpus frequency of ≤ 5 are discarded and the remaining words are ranked and top five words are selected to be added to the category. This process repeats with a larger set of known category members as many number of times as needed. Finally, a user must review the ranked list and judge the words that are true category members. (Riloff and Shepherd 1997) report that it takes human judges 10-15 minutes, on average, to judge the top 200 words for each category which results in approximately 60 valid category members.

(Yarowsky 1992) uses a similar mutual-information-like score to extract *salient* words for a given semantic category in Roget's International Thesaurus:

$$score(w\ for\ Cat) = log\frac{P(w|Cat)}{P(w)} \quad \text{Eq. 3}$$

Where P(w|Cat) is the probability of a word appearing in the contexts of the category members. However, (Yarowsky 1992) does not aim to extract new category members, but the words which are likely to co-occur with the members of the category to be used for Word-sense Disambiguation (WSD). Finally to disambiguate word senses, wherever any of the salient words appears in the context of an ambiguous word, the category for which there is more evidence is detected for the word:

$$\underset{Cat}{Argmax} \sum_{\substack{w\ in \\ context}} score(w\ for\ Cat) \quad \text{Eq. 4}$$

(Turney 2001) used PMI based on frequency counts drawn from the Web. He exploited AltaVista's $hits(query)$ function which returns the number of documents found by the search engine for a given query. He also investigated using AltaVista's $NEAR$ operator which constrains the search to documents that contain the two terms within ten words of one another regardless of the order.

He also proposed a rather different formula for PMI when context words exist, as shown in Eq. 5 (Terra and Clarke 2003). Although scalable to multiple context words, (Turney 2001) argues that each additional context word narrows the sample size making the score more sensitive to noise; therefore, he used only context word in his experiments. He concluded that $NEAR$ co-occurrence statistic along with one context term gives the best results.

$$PMIC(w_1, w_2; C) = \frac{P(w_1, w_2, C)}{P(w_1, C)P(w_2, C)} \quad \text{Eq. 5}$$

2.1.1.2 Hypothesis Testing:

PMI-like estimates are subject to overestimation when the frequencies involved are small. An alternative is Hypothesis Testing measures, which test a null hypothesis $H0$ that co-occurrence of two words is random and there is no association beyond chance. In other words they "measure how surprising the given pattern of co-occurrence would be if the distributions were completely random" (Roark and Charniak 1998). For instance, it is odder for two words having random distributions to occur forty times each and co-occur twenty times in a billion word corpus than each occur only twice but always co-occur. Mutual Information fails to capture this fact.

2.1.1.2.1 Pearson's χ^2

Pearson's Chi-square statistic, described in (Manning and Schütze 1999), determines a specific way to calculate the difference between observed frequencies of two words and expected frequencies under independence assumption. If the difference between the observed and expected frequencies is large, then the null hypothesis of independence can be rejected. The χ^2 is calculated using the following equation (Terra and Clarke 2003):

$$\chi^2 = \sum_{x \in W_1} \sum_{y \in W_2} \frac{\left(f_{x,y} - E_{x,y}\right)^2}{E_{x,y}} \quad \text{Eq. 6}$$

Where $f_{x,y}$ is the observed frequency estimate in the contingency table and $E_{x,y}$ is the expected value under independence assumption. χ^2 test needs a sufficiently large sample size to yield accurate statistics.

2.1.1.2.2 Likelihood ratio (λ)

(Dunning 1993) outlines this statistic and claims it works reasonably well on both large and small corpora and also allows comparison of the significance of both rare and common phenomena. He used a likelihood ratio to test word similarity under the assumption that the words in text have a binomial distribution. He reports improved statistical results versus Pearson's χ^2 test in smaller size texts where the assumption of normal distribution is not valid anymore. Asymptotically, $2\log\lambda$ is χ^2 distributed and is referred to as "Log-likelihood". See (Dunning 1993) for details on how to calculate this metric.

(Roark and Charniak 1998) improve upon (Riloff and Shepherd 1997) using the same generic structure for semi-automatic lexicon induction. They claim the probability score used in (Riloff and Shepherd 1997) is not suitable for generating the final ranking since it favors low-frequency words and that is why a frequency cutoff is used there. They propose using the Log-likelihood statistic for that purpose but keep using a very similar PMI-like score for extracting the candidates:

$$Score(W, C) = \frac{freq(W\ in\ C's\ contexts)}{freq(W\ in\ any\ word's\ contexts)} \quad \text{Eq. 7}$$

They believe this score is conservative enough to select good candidates as opposed to the Log-likelihood which tends to favor high-frequency candidates. High-frequency terms have a broad coverage in their contexts and, potentially, can cause a drastic drift from the category in subsequent iterations if they are not true category members.

(Roark and Charniak 1998) also investigate the problem of selecting initial seeds which was not addressed in (Riloff and Shepherd 1997). They conclude that, since the PMI score will tend not to select higher frequency words in the category, initial seeds should be among the most frequent words in the corpus to provide the broadest coverage of category occurrences from which additional likely category members will be selected.

2.1.2 Syntactical Co-occurrence Models

In general, Co-occurrence models do not necessarily yield tightly associated words. Even if they do, not every two tightly associated words belong to the same semantic category (e.g., "Libya" and "Gadaffi"). Co-occurrence models use no evidence whatsoever to ensure that constraint. Given a Part Of Speech (POS) tagged or parsed corpus, interesting syntactical structures can be used to accurately obtain semantic

knowledge about words. These structures can be used independently or in combination with statistical methods by, let's say, restricting makeup of text windows to focus on specific relationships between words.

However, not only tools such as parsers and POS taggers are error-prone, exploiting syntactical information to learn semantic classes of words is not straightforward. According to (Phillips and Riloff 2002):

1. Lexico-syntactic expressions that explicitly indicate semantic relationships (e.g., lists in the form of "NP, NP, and other NPs") are reliable but a lot of semantic information occurs outside these expressions.
2. General syntactic structures (e.g., lists and conjunctions) capture a wide range of semantic relationships. For example, conjunctions frequently join items of the same semantic class (e.g., "cats and dogs"), but they can also join different semantic classes (e.g., "fire and ice").

A good syntactical model for the task of semantic lexicon induction is one which is both exhaustive and specific; meaning that it captures a wide range of semantic relationships which are specific to members of a semantic class. Here we aim to cover different syntactical models used in this task.

(Roark and Charniak 1998) criticize the assumption made by (Riloff and Shepherd 1997) that members of the same semantic category would co-occur in discourse. They find such a window based co-occurrence more suitable for word sense disambiguation where words provide disambiguation characteristics regardless of their part-of-speech or semantics. Thus, they focus exclusively on certain syntactic structures: conjunctions, lists, and appositives (they consider all list elements co-occurring with one another).

Similarly, (Caraballo 1999) focuses only on conjunctions and appositives and (Widdows and Dorow 2002) on symmetric "noun and/or noun" patterns (conjunctions). Such symmetric patterns can be easily represented by graph models to infer semantic similarity. (Widdows and Dorow 2002) exploit a graph model to build a semantic lexicon form a number of seeds in an iterative approach: each node represents a noun and each edge represents two co-occurring nouns separated by conjunctions *and* or *or*. Let A be a set of nodes, $N(a)$ be set of a's neighbors and $N(A) = \bigcup_{a \in A} N(a)$. They introduce an *affinity* score (Eq. 8) between node u and set A and use it to rank and select new extractions.

$$\frac{|N(u) \cap N(A)|}{|N(u)|}$$ Eq. 8

For evaluation, they pick 10 WordNet categories and retrieve 20 words given one seed for each category. They propose, but do not evaluate, using more than one seed to start with to avoid infections arising from bogus co-occurrences (e.g., idioms) and ambiguity (e.g., "Apple" could be a fruit or a company). Out of 200 extractions, they report an accuracy of 82% and compare it to 17% and 35% reported by (Riloff and Shepherd 1997) and (Roark and Charniak 1998) respectively. However, one could question the reliability of these comparisons for a number of reasons:

1. Each of those two algorithms extracted approximately between 200-260 terms per category. A *category drift* tends to be observed in later iterations. Evaluating on 20 extractions does not show the method's performance in preventing the drift.
2. In the other two approaches, the evaluation is done on the MUC-4 corpus which is 200 times smaller in size than the British National Corpus used by (Widdows and Dorow 2002).
3. Evaluations in other two approaches are manually and therefore subjective and non-uniform.

Such graph models can also be used with graph clustering algorithms for unsupervised semantic clustering of words. (Dorow, et al. 2005) build a graph representation of words appearing in conjunctions and use two approaches to soft clustering[2] of words in the graph. (Davidov and Rappoport 2006) use a similar graph model while focusing only on symmetric lexical relations of the form <candid>H<candid> or <candid>HH<candid> where H is any high frequency word appearing more than T_H times per million words. Their rationale is that two content-bearing words that appear in a symmetric pattern are likely to be semantically similar in some sense. They propose a method to discover these patterns by graph-theoretic measures and identify semantic categories using a graph clique-set algorithm in a fully unsupervised manner.

[2] In soft clustering, items can belong to more than one cluster.

2.1.3 Contextual Similarity Models

As stated before, the underlying assumption of the co-occurrence statistics that semantically similar words tend to co-occur text is a weak assumption and results in loosely related words to be falsely detected semantically similar. Using syntactical clues tends to improve the precision of co-occurrence models by helping them focus on candidates which are more likely to be of the same semantic class. However, they are still restricted as they need members of the semantic category co-occur in text. In these models, many undesirable semantic associations are captured too. In fact, "statistical models are still being used to separate the meaningful semantic associations from the spurious ones" (Phillips and Riloff 2002).

An alternative to co-occurrence models is the models based on the fundamental hypothesis that "words that occur in the same contexts tend to have similar meanings" (Harris 1954) which is known as the *Distributional Hypothesis*. In other words, the more inter-substitutable the words are in text, the higher the chance they belong to the same semantic class.

To measure similarity of the contexts words appear in, first we need to define "context". Context can take different meanings and representations, but more or less falls into three broad categories:

1. A window of words/tokens around the category member.
2. Grammatical relations (e.g., subject-verb or object-verb) between the category member and other words in the sentence (D. Lin 97).
3. Lexico-syntactic templates in which members of a semantic class are expected to appear.

Different types of contexts vary in their relative power in capturing semantic classes of words, genre- or domain-independency, extraction power (coverage), and finally applicability in terms of cost of construction.

There are two main paradigms in the literature for using contexts of words to capture semantic information. The first group of methods relies on ***vectors of contexts*** to infer semantic similarity and the second one uses them as ***extraction patterns*** to discover semantically similar words. We introduce both trends and different definitions of context they utilize.

2.1.4 Context Vector Models

In these methods terms are represented as vectors of contexts with which they appear in text. Regardless of type of context being used, they use some sort of similarity measure between these context vectors which is, hypothetically, indicative of semantic similarity between the given terms.

(Caraballo 1999) creates a vector for each noun containing all the nouns it appears in a conjunction or appositive with, along with the respective frequency counts. She measures the similarity of the vectors for two nouns by computing the cosine of the angle between the vectors. These similarity scores are used in a bottom-up clustering method to build clusters of semantically similar nouns.

(D. Lin 98) proposes a similarity score between w_1 and $w_2$based on the commonality of the dependency triples the two words appear in, where a dependency triple $(w\ r\ w')$ means an asymmetric grammatical relationship between w and w' in a sentence. For example the dependency triples in the sentence "I had a delicious pizza" are:

(had *nsubj* I), (pizza *det* a), (pizza *amod* delicious) (had *dobj* pizza)

For each word w, (D. Lin 98) forms a vector of all the dependency triples it occurs in along with their respective frequency counts ($\|triple\|$). He defines the information contained within $\|w_1\ r\ w_2\| = c$ as follows:

$$I(w_1\ r\ w_2) = \frac{\|w_1\ r\ w_2\| \times \|*\ r\ *\|}{\|w_1\ r\ *\| \times \|*\ r\ w_2\|} \qquad \text{Eq. 9}$$

Where * means a wildcard. (D. Lin 98) states that $I(w_1\ r\ w_2)$ correspond to the mutual information between w_1 and w_2. If $T(w)$ is the set of all the pairs $(w'\ r)$ for which $I(w\ r\ w') > 0$, he defines the similarity between w_1 and w_2 as follows:

$$\frac{\sum_{(r\ w)\in T(w_1)\cap T(w_2)}\big(I(w_1\ r\ w) + I(w_2\ r\ w)\big)}{\sum_{(r\ w)\in T(w_1)} I(w_1\ r\ w) + \sum_{(r\ w)\in T(w_2)} I(w_2\ r\ w)} \qquad \text{Eq. 10}$$

In addition to his proposed similarity measure, he also investigates using other commonly used similarity measures such as sim_{cosine}, sim_{dice}, sim_{jacard} (Frakes and Baeza-Yates 1992).

(Lin and Pantel 2002) introduce CBC (Clustering By Committee) for clustering of semantically similar words. They use the same features, weights, and similarity score as in (D. Lin 98). The centroid of a cluster is constructed by averaging the feature vectors of a subset of the cluster members (committees). In

order to form the committees, they compute pairwise similarities between words that share high mutual information features to reduce the quadratic complexity of computing the complete similarity matrix between pairs of elements.

(Pantel and Ravichandran 2004) search centroids of the clusters for particular features such as *Apposition (N:appo:N)*, *Nominal subject (N:subj:N)*, *Such as (N:such as:N)*, and *Like (N:like:N)*; for every cluster, they sum up the mutual information scores for every term that participates in those relationships with a committee member and choose the highest scoring term as the name of the class.

Vechtomova and Robertson (Vechtomova and Robertson 2011) use Minipar[3] (D. Lin 1993) to extract dependency triples form text and form feature vectors for candidate entities and a set of seed words. To calculate the similarity between a candidate entity *c* and a seed *s*, they adapt BM25 with query weights (Robertson, et al. 1995) and calculate a Query Adjusted Combined Weight (QACW) (Spärck Jones, Walker and Robertson 2000) treating the vector of the seed as the query and the vector of the candidate as the document:

$$QACW_{c,s} = \sum_{f=1}^{F} \frac{TF(k_1+1)}{k_1 \times \left(\frac{(1-b)+b \times DL}{AVDL}\right) + TF} . QTF . IDF_f \qquad \text{Eq. 11}$$

Where F is the number of features c and s have in common; TF, the frequency of feature f in the vector of c; QTF, the frequency of feature f in the vector of s; k_1 and b, normalization factors; DL, length of vector of c; $AVDL$, average length of vectors of all candidate entities; and IDF_f is IDF of a feature. For each entity they calculate sum of this score for all the seeds multiplied by the seed's $TFIDF$ score. They also incorporate entity's $TFIDF$ and rank them accordingly. (Vechtomova and Robertson 2011) demonstrate that their method performs better on Entity track's Related Entity Finding (REF) dataset of TREC 2010 and Question Answering (QA) dataset of TREC 2005, while Lin's method (D. Lin 98) does a good job of filtering out non-relevant entities.

Some works use shallow lexical features instead of grammatical dependencies (Fleischman and Hovy 2002) and (Pantel, Crestan, et al. 2009). (Pantel, Crestan, et al. 2009) record term contexts in a large collection of unstructured text and construct a term-context matrix. Terms are Noun Phrases (NP) and

[3] http://webdocs.cs.ualberta.ca/~lindek/minipar.htm

features are defined as the rightmost and leftmost stemmed NPs. They weight each feature using PMI between the term and the feature and ultimately use Euclidean Distance, Cosine, and Dice measures to calculate similarity between two context vectors. They take as input a list of seeds and rank all the terms according to their similarity to the centroid of seed list. Due to large size of the corpus, they use the map-reduce framework along with some other heuristics to make computations possible.

(Cimiano and Volker 2005) compare windows of words against pseudo-syntactic dependencies as the features. The pseudo-syntactic dependencies are not extracted from dependency parse trees as in (D. Lin 98), but through applying regular expressions to POS tagged text (see Table 3). Their rationale for using pseudo-syntactic dependencies is to achieve less sparse context vectors based on the observation in (Grefenstette 1994) that "the quality of using word windows or syntactic dependencies for distributional analysis depends on the frequency of the word in question".

Dependency type	**Example**
adjective modifiers	"a nice city" → nice(city)
prepositional phrase modifiers	"a city near the river" → near river(city) and city near(river)
possessive modifiers	"the city's center" → has center(city)
noun phrases in subject or object position	"the city offers an exciting nightlife" → offer subj (city) and offer obj(nightlife)
prepositional phrases following a verb	"the river flows through the city" → flows through(city)
copula constructs	"a flamingo is a bird" → is bird(flamingo)
verb phrases with the verb *to have*	"every country has a capital" → has capital(country)

Table 3: Pseudo-syntactic dependencies used by (Cimiano and Volker 2005)

They observe window-based context perform slightly worse than pseudo-syntactic dependencies for classification of named entities as well as make context vectors larger, making the similarity calculation less efficient.

2.1.5 Pattern-based Models

2.1.5.1 Domain-independent patterns

Hyponymy patterns, introduced by (Hearst 1992), are generic hand-crafted patterns that capture semantic association between words by looking for explicit hyponymy relations in text. The main idea is that if a term *t* belongs to a class *c*, we should expect occurrences of phrases like "such *c* as *t*". (Hearst 1992) proposed six domain-independent lexico-syntactic templates that capture a hyponymy relation between a semantic class and a word. List of the templates she used is presented in Table 4. She applied the patterns to text and validated the extracted hyponyms against WordNet. She also proposed a method to augment WordNet using them.

Template	**Example**
NP such as {NP,} {(or|and)} NP*	The bow lute, such as the Bambara ndang, is plucked.
such NP as {NP,} {(or|and)} NP*	Works by such authors as Herrick, Goldsmith, and Shakespeare
NP {, NP}{,} or other NP*	Bruises, wounds, broken bones or other injuries
NP {, NP}{,} and other NP*	temples, treasuries, and other important civic buildings.
NP{,} including {NP ,} {or | and} NP*	All common-law countries, including Canada and England
NP {,} especially {NP,} {or | and} NP*	most European countries, especially France, England, and Spain

Table 4: Hearst's templates for hyponym acquisition

In fact, Hearst initially came up with patterns 1 and 3 and used the following method to discover the rest (Hearst 1992):

1. Provide a list of pairs that hold a certain hyponymy relation (e.g., "England-country").
2. Find places in the corpus where the terms occur syntactically near one another and record the environment.

3. Manually find the commonalties among these environments and once a new pattern has been positively identified, use it to gather more instances of the target relation and go to Step 1.

She claims that these patterns are frequent and accurate (they almost all the time point out the desired relationship) and can be applied to plain texts of any genre. (Kozareva, Riloff and Hovy 2008) show that despite the highly restrictive nature of these patterns, they still produced many incorrect instances. They propose a graph model that represents the links between hyponym occurrences in these patterns and increase the bootstrapping precision to a great extent.

Although allegedly frequent, these patterns do not appear frequently enough in domain-specific corpora to extract rare and domain-specific terminology. For this reason, some approaches use these patterns on the Web instead (Vechtomova and Robertson 2011). Vechtomova et al. use Hearst's hyponym acquisition method to discover entities that belong to the semantic category in question for the Related Entity Finding (REF) task of the Entity track of TREC 2010 (Balog, Serdyukov and de Vries 2010).

(Vechtomova and Robertson 2011) propose a method to extract category names automatically from the REF task's topic narratives, create six queries based on each of Hearst's templates, and submit to a commercial search engine to retrieve instances (e.g., "recording companies such as" where "recording companies" is the category name). In presence of enough number of instances, Hearst's method seems to be applicable to fine-grained semantic categories (e.g., "formula one drivers" or "operating systems"). Of course too narrow of a category can naturally yield very few to no hyponymy relation instances.

(Paşca 2004) applies the following hyponym pattern inspired by (Hearst 1992) to the Web to learn members of semantic classes:

$$\langle [startOfSent]\ X\ [such\ as|including]\ N\ [and|,|.]\rangle$$

Where N is the potential instance and X is the category name.

The KnowItAll system introduced by (Etzioni, et al. 2005) uses the set of generic extraction patterns introduced by (Hearst 1992) as well as "NP is a <class>" and "NP is the <class>" to capture the "is-a" relationships. KnowItAll extracts class instances from the Web and further validates them by computing a PMI score between each instance and multiple automatically generated discriminator phrases associated with the class (e.g., "X is a city" for the class city). The PMI is in fact the number of Search Engine hits for a query that combines the discriminator and instance, divided by the hits for the instance alone. These scores are then used to classify the instances as true or false.

(Phillips and Riloff 2002) also try to capture membership in a semantic class by looking for words that have the same hypernym which are, as they phrase, *semantic siblings* (e.g., "dog" and "frog" both have the hypernym *ANIMAL*). They propose a bootstrapping algorithm that uses syntactic structures in combination with heuristics to extract noun phrases of the same semantic class with high precision:

1. Appositives: is a syntactic structure that consists of a noun phrase (NP), followed by a comma, followed by another NP, where the two NPs have the same reference. They only use appositives that contain one proper noun phrase and one general noun phrase (G*NP, PNP*) such as "President, Barak Obama".
2. Compound Nouns in the form of "GN PNP" such as "president Barak Obama".
3. ISA Clauses: an NP followed by a VP that is a form of "to be", followed by another NP (PNP + "to be" verb + GNP). For example "Barak Obama is the president".

For each category (e.g., "People"), the bootstrapping framework populates a lexicon of general nouns (GN) such as "president" and a second lexicon of proper nouns (PNP) such as "Barak Obama" simultaneously using the above syntactic patterns with minimal use of statistics. They also combine the three classifiers using a co-training model that increases recall while maintaining almost the same precision.

Since their bootstrapping approach highly depends on simultaneously populating PNP and GN lexicons to work, one might raise doubt about its performance on narrower categories such as "house DJ" where there is not much room for building lexicons of general nouns. However, (Phillips and Riloff 2002) do not investigate this and evaluate their approach on three broad categories of "People", "Organization", and "Product".

Although high precision, these types of syntactic structures (Hyponym-hypernym patterns) can extract only those members of the semantic category that are explicitly found along with general nouns (or the semantic class label). Obviously this is not always the case in discourse and as a result these patterns tend to sacrifice recall in favor of precision.

2.1.5.2 Domain-specific patterns

While generic extraction templates (e.g., hyponymy patterns) extract members of a desired semantic class with high precision, there are only a handful of such manually-generated patterns out there and also most

of the best extraction rules for a domain do not match the generic ones (e.g., "I worked at <COMPANY> for…"). On the contrary, we can have many domain-specific patterns that are able to capture this information precisely. The substantial amount of previous work in semantic lexicon induction, named-entity recognition (NER), relation extraction (RE), and question answering (QA) tasks reflects the power of domain-specific patterns to obtain semantic information about words.

Inspired by Hearst's Pattern Learning scheme (Hearst 1992), numerous methods have been proposed to automatically learn extraction patterns from text. These methods are usually semi-supervised and start with a set of seeds, either given by the user or extracted via domain-independent patterns. They look for instances of the seeds in text and learn useful patterns that tend to co-occur with other category members as well.

Patterns used in semantic lexicon induction and NER tasks are singly anchored since they are supposed to extract one category member and the ones used in QA or RE are doubly anchored since their job is to capture a relation between two terms (see Table 5). Although specific representation of patterns varies across these tasks, they are used in almost the same way and for the same purpose of capturing the semantic knowledge about words. For this reason, we cover the previous work in those threads of research as well:

Task	**Pattern Example**
Semantic lexicon induction (find instances of *PRESIDENT*)	<PRESIDENT> was elected as president in 1986
Question answering (when was Roosevelt born?)	<PERSON> was born in <DATE>
Relation finding (where is Microsoft's headquarters?)	<COMPANY> is headquartered in <CITY>

Table 5: Examples of domain-specific patterns used in different tasks

Lexical patterns are domain-specific patterns that are some representation of neighboring tokens of terms, usually in the form a window (or vector) of tokens appearing before and after the category members, sometimes obtained using heuristics and syntactic clues (lexico-syntactic patterns). (Ravichandran and Hovy 2002) learn surface lexical patterns from Web documents to answer questions such as "when was X born?" where the question and answer can be represented by a pair of words (e.g., <X, 1948>). These shallow patterns are computationally cheap to obtain since there is no need for expensive preprocessing of text such as parsing.

They submit a query made up of one example question and answer pair to a search engine. Among the top retrieved documents, they extract, as patterns, substrings of any length (limited to sentence boundaries) which contain both question and target term. They further calculate the likelihood of a pattern extracting the correct answer by retrieving the top documents from the Web, this time omitting the answer term, and observe what percentage of the matches contain the correct answer. Patterns that extract the correct question-answer pair half of the time and have high average precision scores are stored in a table for a given question type. Every time a question of the same type is encountered, these patterns will be applied to Web documents to extract answers which are sorted by patterns' precision scores.

1.0	<QUESTION>(<ANSWER> -)
0.85	<QUESTION> was born on <ANSWER>
0.6	<QUESTION> was born in <ANSWER>
…	…

Table 6: Extraction patterns discovered by Ravichandran's approach

(Paşca 2004) proposes re-applying pairs of <category, instance> learned via domain-independent patterns to the text and capture the text around them to acquire a more diverse set of domain-specific patterns.

(Etzioni, et al. 2005) extend their KnowItAll system by enabling it to learn lexical patterns and observe a significant increase in the number of sentences from which facts are extracted. After generating a ranked list of seeds, a query is generated for each seed and submitted to a Web search engine and context strings around instances of the seeds (w words before and after) are captured. Patterns are defined as substrings of a context string with the seed instance replaced by a placeholder. Patterns are then ranked by their precision (the fraction of entities it extracts that belong to the target class) and recall (the fraction of target class members extracted by it). They use these patterns to extract more instances and add the top extractions to the target class. This process can be repeated in a bootstrapping manner.

(Agichtein and Gravano 2000) developed the "Snowball" system, inspired by DIPRE (Brin 1998), that extracts <organization, location> pairs from a named entity tagged corpora. It takes a small set of seed pairs as input, and to generate extraction patterns it looks in the regions in the text corpus where the elements of the seed pairs occur close to each other. Instead of a window of words, it represents the patterns in a more flexible way so that they have more coverage while still being selective. A snowball pattern is a 5-tuple $\langle \text{left}, \text{ORG}, \text{middle}, \text{LOC}, \text{right} \rangle$ where right r, left l, and middle m are vectors of terms

appearing in windows of a certain length to the left and to the right of the entity pair. Those 5-tuples of a given seed that are similar enough would form a centroid 5-tuple where each term is weighted according to its frequency in the corresponding context. Each vector is also multiplied by a scaling factor which implies its relative importance (e.g., middle vector has higher importance). For extracting new pairs, Snowball associates an equivalent 5-tuple with each document section that contains two named entities with correct tags and calculates a similarity score in the following way:

$$match(c_1, c_2) = l_{c_1}.l_{c_2} + m_{c_1}.m_{c_2} + r_{c_1}.r_{c_2}$$

(Agichtein and Gravano 2000) score a pattern according to its confidence which is a function of its selectivity. To account for its coverage they use the *RlogF* metric proposed by (E. Riloff, Automatically generating extraction patterns from Untagged Text 1996). *RlogF* confidence of a pattern is high if a large proportion of its extractions are correct or it extracts a lot of correct pairs:

$$conf_{RlogF}(P) = \frac{P.positive}{P.positive\ \times P.negative} \times log_2(P.positive) \qquad \text{Eq. 12}$$

An extraction is positive if the <organization, location> pair is correct according to previous iterations and negative otherwise. The $conf_{RlogF}$ scores are then normalized and treated as the probability of a pattern extracting valid tuples. Having these probabilities and assuming they are independent, the probability that a tuple is valid is calculated as follows:

$$Conf(T) = 1 - \prod_{i=0}^{|P|}\big(1 - Prob(P_i).Match(C_i, P_i)\big) \qquad \text{Eq. 13}$$

Where $P = \{P_i\}$ is the set of patterns that extract tuple T and C_i is the context associated with an occurrence of T that matched P_i. High confidence tuples are then added to a knowledge base as valid pairs and bootstrapping continues.

(Welty, et al. 2010) use simple lexical patterns to extract four types of <actor, show>, <author, written-work>, <director, movie>, and <parent, child> relations from texts with the help of two knowledge bases (IMDB and Freebase) that contain a large set of tuples having these relations. To determine the degree of association of a pattern to a certain relation, they propose the following method to gather statistics, although they do not suggest any learning algorithm to use that data:

First, each pattern is associated with all the relations whose tuples are extracted by that pattern. For every relation a tuple participates in, the "positive count" of the corresponding relation in the pattern is

incremented and, similarly, for every relation in the pattern that does not exist in the KB for the tuple, their "negative count" is incremented.

They also make some important observations:

1. The frequency of patterns that express a particular relation has a very long tail (see Figure 1), and only a large number of seeds can make learning infrequent, but maybe effective, patterns possible.
2. Many of the patterns induced depend heavily on typing and context to work (e.g., use parenthesis or other punctuation to signal the relation). These patterns are generally unreliable for detecting relations unless types of tuples are known in advance (e.g., "actor-name (movie-name)" pattern in "DiCaprio (Titanic)").
3. Pattern generalization has a significant impact on recall with negligible negative impact on precision. For example, "<actor> appeared in the 1994 screen epic <movie>" can be generalized to "<actor> appeared in the <date> screen epic <movie>" without any impact on the precision of the pattern.

(Welty, et al. 2010) also try patterns based on the spanning tree between the two arguments in a dependency parse, but provide neither further details on how they were constructed, nor evaluation results.

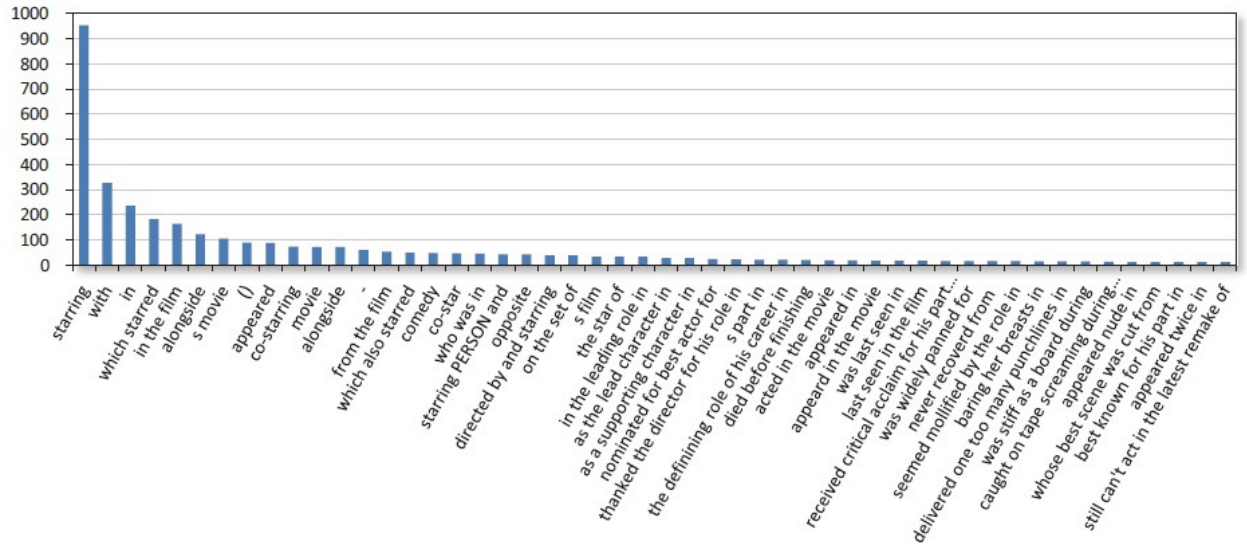


Figure 1: Frequency distribution of patterns (Welty, et al. 2010)

(Whitelaw, et al. 2008) use a supervised model to predict the entity type as a function of features of a mention. They use lexical patterns not to extract entities, but to use as a filter to retain reliable entities for training a classifier. The classifier uses features such as the URL of the Webpage the mention appears in, words composing the entity, and each of the three tokens immediately before and after the term (position-specific). To obtain patterns of filtering, they submit a very large set of seed entities to the web pages and for every occurrence of each seed, extract three tokens from left and right of it. These text windows along with all shorter substrings of them are recorded and those which have the following characteristics are kept as patterns:

1. *Type Specific*: at least 80% of its extractions belong to the same type
2. *Frequent*: have at least 50 occurrences
3. *Source diverse*: How many different sources (hosts, domains, etc.) a template was found in
4. *Name diverse*: the more different names appear in a template, the more likely it is that other good names will appear in it.
5. *Have a proper makeup*: having at least four tokens and being at least 8 characters long.

The above criteria are chosen to be conservative to give the most reliable patterns to subsequently retain the most trusted entities for training the classifier. Some works go beyond selecting patterns by a measure of their selectivity and coverage. Instead, they use (semi-) supervised approaches to learn the most effective patterns.

(Banko and Etzioni 2008) consider each pair of noun phrases appearing no more than a maximum number of words apart and their surrounding contexts are regarded as possible evidence for RE. The system detects relations by using Conditional Random Fields (CRF) to label the context of the two candidate noun phrases (Figure 2). CRFs (Lafferty, McCallum and Pereira 2001) are undirected graphical models trained to maximize the conditional probability of a finite set of labels Y, given a set of input observations X.

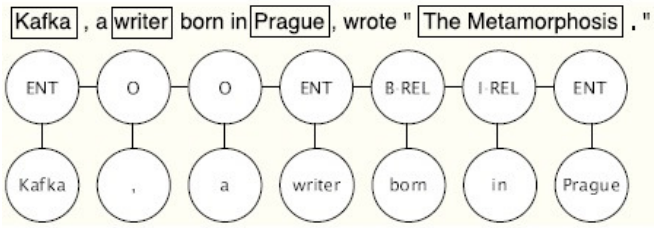


Figure 2: Relation Extraction as Sequence Labeling: a CRF is used to identify the relationship, *born in*, between *Kafka* and *Prague* (Banko and Etzioni 2008)

The sets of features include part-of-speech tags, capitalizations, punctuations, and context words occurring within six words to the left and to the right of the current word. Although (Banko and Etzioni 2008) propose an Open Relation extractor and do not try to capture semantic classes of words, the CRF model they use is applicable to the latter task as well.

(E. Riloff 1996) introduce AutoSlog that automatically generates extraction patterns using syntactical heuristic rules. As input, AutoSlog takes a training corpus in which desired NPs are labeled, parses the text using a shallow parser and applies heuristics to obtain extraction patterns. AutoSlog uses the following domain-independent heuristic rules that are likely to capture the semantic class of the NPs:

Pattern	Example
<subj> passive-verb	<victim> was murdered
<subj> active verb	<prep> bombed
<subj> verb infin.	<prep> attempted to kill
<subj> aux noun	<victim> was victim
passive-verb <dobj>	killed <victim>
active-verb <dobj>	bombed <target>
infin. <dobj>	to kill <victim>
verb infin. <dobj>	tried to attack <target>
gerund <dobj>	killing <victim>
noun aux <dobj>	fatality was <victim>
noun prep <np>	bomb against <target>
active-verb prep <np>	killed with <instrument>
passive-verb prep <np>	was aimed at <target>

Table 7: AutoSlog Heuristics

Since patterns generated in this way can be undesirable, a person should manually go through them to accept or reject them.

Manual labeling of a corpus of 1000 texts takes approximately one week (E. Riloff 1996). It is also a non-trivial task and needs a domain expert. Riloff later developed AutoSlog-TS (E. Riloff 1996) to avoid these problems. AutoSlog-TS takes a classified corpus of relevant and non-relevant texts for the domain; after applying the heuristics to the texts and generating the patterns, AutoSlog-TS computes relevance statistics for each pattern by estimating the conditional probability that a text is relevant given that it activates a particular extraction pattern (Eq. 14).

$$relevance\ rate = P(relevant\ text | text\ activates\ pattern_i) = \frac{rel - freq_i}{total - freq_i} \quad \text{Eq. 14}$$

Where $rel - freq_i$ is the number of times $pattern_i$ was activated in relevant text, and $total - freq_i$ is the total number of times $pattern_i$ was activated in the entire corpus. The rationale behind the conditional probability is that domain-specific patterns tend to appear substantially more often in relevant texts than in non-relevant ones. Finally, AutoSlog-TS ranks patterns according to their frequency and relevance score:

$$relevance\ rate \times log_2(frequency) \quad \text{Eq. 15}$$

AutoSlog-TS demonstrates better precision in comparison to AutoSlog, but falls behind in recall. Riloff (E. Riloff 1996) observe that many useful patterns are buried deep in AutoSlog-TS's ranked list, which cumulatively could have a substantial impact on performance. She argues that the current ranking formula is biased towards high-frequency patterns and a better ranking scheme might be able to balance high-frequency and low-frequency, but effective, patterns. She also proposes adding semantic constraints to the lexico-syntactic patterns for increased accuracy (e.g., in "killed <victim>" <victim> should be checked to be PERSON)

Meta-Bootstrapping (Riloff and Jones 1999) and Basilisk (Thelen and Riloff 2002) are two bootstrapping algorithms for semantic lexicons induction which use the extraction patterns introduced in AutoSlog.

2.1.5.2.1 Meta-Bootstrapping

It starts with a set of seed words and an unannotated domain-specific text corpus. Before bootstrapping begins, AutoSlog is run exhaustively over the corpus to generate extraction patterns for every noun phrase. This results in thousands of extraction patterns. Meta-Bootstrapping scores these patterns using the *RlogF* metric introduced in (E. Riloff 1996). *RlogF* is a metric designed for Information Extraction tasks and tries to make a balance between reliability and frequency:

$$score(pattern_i) = \frac{F_i}{N_i} \times log_2(F_i) \qquad \text{Eq. 16}$$

Where N_i is the total number of unique noun phrases $pattern_i$ extracted and F_i is the total number of unique lexicon entries among extractions of $pattern_i$. The best pattern is saved and all its extractions are labeled as members of the semantic category. This process, called *mutual bootstrapping*, is repeated and the scores for patterns are recalculated taking into consideration the newly labeled words. It is logical to think that not all the extractions of a pattern might truly belong to the category, so adding all the extractions by a pattern to the lexicon can rapidly infect it in early stages of bootstrapping.

(Riloff and Jones 1999) add a second level of selection criteria (*meta-bootstrapping*) to increase the robustness of the algorithm. In fact, they re-evaluate the extractions of the top pattern and select only the top most reliable ones to be added to the semantic lexicon. The basic intuition is that the more patterns the NP co-occurs with, that more it is likely to belong to the category, simply because there is more evidence for its membership in the category (see Eq. 17). They also introduce a small factor reflecting the strength of the patterns into the formula mainly for tie-breaking purposes.

$$score(NP_i) = \sum_{k=1}^{N_i} 1 + \left(.01 \times score(pattern_k)\right) \qquad \text{Eq. 17}$$

2.1.5.2.2 Basilisk

Introduced in (Thelen and Riloff 2002), Basilisk starts with an unannotated text corpus and a small set of seed words. As opposed to Meta-Bootstrapping that trusts all the extractions of a single best pattern for addition to the semantic lexicon, Basilisk scores an extraction based on combined evidence from *all* patterns that extract a term.

Basilisk generates and ranks the extraction patterns in a way identical to Meta-bootstrapping. It selects the best ones into the *Pattern Pool* and then places all the head nouns of the NPs extracted by those

patterns in the *Candidate Word Pool*. Entries in the Candidate Word Pool are scored based on the collective reliability of *all* the patterns that extracted them (Eq. 18). Top five nouns are labeled and added to the lexicon:

$$AvgLog(word_i) = \frac{\sum_{j=1}^{P_i} log_2(F_j + 1)}{P_i} \quad \text{Eq. 18}$$

Where P_i is the number of patterns that extract $word_i$ and F_j is the number of unique category members extracted by pattern j. Using this formula, "a word receives high score if it is extracted by patterns that tend to extract known category members" (Thelen and Riloff 2002). The log is used to prevent bias towards patterns that extract a large number of category members. High-frequency patterns may extract many existing category members, but they can be loosely associated with the category and extract many valid entries too.

In comparative experiments by (Thelen and Riloff 2002), Basilisk shows better performance than Meta-Bootstrapping on a number of categories. It incorporates the second level of bootstrapping proposed in Meta-Bootstrapping into the algorithm. To score the candidate entries, it takes into account reliability of the patterns (their association with the lexicon), while Meta-bootstrapping uses patterns' scores for tie breaking only. Basilisk also has the advantage that it considers extractions of a set of top performing patterns for addition to the lexicon, rather than extractions of one single best pattern.

2.2 Supervised Approaches

There a few works on supervised approaches to building semantic lexicons. These approaches are close in nature to Named-Entity Recognition (NER) systems that take a block of text and classify the elements into predefined categories such as PERSON, ORGANIZATION, LOCATION, etc. Powerful NER systems (McCallum and Li 2003) train statistical sequential labeling models such as CRF using an array of features (e.g., is the word capitalized?, is it followed by "Inc."?, or does it appear in a lexicon of company names?) to label named-entities in text. CRFs are good at ambiguity resolution and generally outperform rule based systems by large margins (rules can be incorporated as features into CRFs).

(Tsai and Chou 2011) develop a method to extract dish names from Chinese review texts. They efficiently identify candidate dishes by constructing a suffix array for each restaurant (from its reviews concatenated together) and extracting all frequent substrings that appear more than twice. They further

filter these candidates using heuristics based on their POS tags and makeup and employ CRF with features such as prefixes and suffixes of words, quotation marks, style (font/color/hyperlink) and image proximity to validate them.

Supervised approaches are usually high precision but low recall due to unfeasibility of building large high-coverage training sets of labeled data. Therefore they use gazetteers (lexicons) of ORGANIZATIONS, PERSONS, etc. to be able to detect non-trivial Named-Entities. This makes sequential labeling models such as CRF less applicable and as a result less popular in the task of automatic semantic lexicon construction where the goal is to inexpensively obtain fine-grained semantic lexicons of a domain-specific vocabulary. Thus, some works propose methods to obtain/expand semantic lexicons using unsupervised approaches and feed those lexicons to Named-Entity Recognizers to achieve higher recall and yet benefit from high precision labeling (e.g., (McCallum and Li 2003) and (Liu, et al. 2011)).

(McCallum and Li 2003) use a technique they call "WebListing" to expand a set of seed words automatically by using lists and tables found on HTML WebPages. They also experiment using Google Sets[4] for this purpose.

Semantic tagging of queries (Figure 3), i.e. assigning a pre-defined semantic label to query terms, is a specialized form of Information Extraction (IE) or Named-Entity Recognition. (Liu, et al. 2011) and (Wang, et al. 2009) use CRF models for this purpose, exploiting automatically expanded lexicons as features. Lexicons are generated from Web search query logs and Web lists using semi-supervised methods.

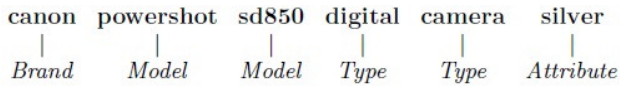


Figure 3: Query Semantic Tagging (Wang, et al. 2009)

[4] The Google Labs experiment to automatically expand a set of words with semantically similar words found on Web was shut down on September 5, 2011.

Chapter 3

Methodology

The works reviewed in the previous chapter as well as many others in related research topics suggest semi-supervised methods, sometimes used along with supervised methods for increased accuracy, to be an effective and inexpensive approach to building semantic lexicons. They also demonstrate the advantages of using contextual evidence to determine semantic class/similarity of words. As stated before, semi-supervised contextual similarity models fall into two categories:

1. *Context Vector Models*: compute pair-wise similarities between candidate terms and seeds in an exhaustive way and rank the terms accordingly. The top m terms are deemed to belong to the category. The complexity of these methods is $O(nk)$, where k is the number of seed words and n is the number of candidate terms to be ranked.
2. *Pattern-based Models*: gradually add the most likely category members to the lexicon (which is initially composed of seed words) in an iterative fashion. They use term contexts not as features to compute similarity, but as patterns/templates to extract semantically similar terms: Given a small number of seeds, they extract patterns with which they co-occur in text and use these patterns to identify other category members. This bootstrapping process can be repeated as many times as needed.

Although nontrivial to compute, Pattern-based Models are computationally less expensive than Context Vector Models: they avoid dealing with a lot of spurious candidates by focusing only on useful patterns that tend to extract category members and practically discarding a subset of the candidate terms.

In this work, we introduce an effective semi-supervised bootstrapping method for automatic construction of semantic lexicons, which has the following characteristics:

1. Uses shallow lexical patterns to capture semantic information about words, making it applicable to large text corpora.

2. Uses no supervision or machine learning method for learning extraction patterns.
3. Has an efficient implementation and applies heuristics to maintain computation costs at the lowest and increase scalability.
4. Learns useful domain-specific extraction patterns while building the semantic lexicon. These patterns can potentially be applied to other corpora to learn category members.

3.1 Research Questions

Besides proposing an effective bootstrapping method, we investigate the following research questions:

1. Can lexical patterns effectively capture information about semantic classes of words or do we need richer linguistic information, such as syntactic roles of words obtained from a deep parser?
2. How do iterative bootstrapping models perform compared to one-pass context vector models?
3. What should be chosen as the extraction unit?
4. How can category drift be prevented in the output semantic lexicon?
5. What are the effects of the number and composition of seeds on bootstrapping performance?

3.2 The Algorithm

Figure 4 shows a high level overview of our semi-supervised bootstrapping model. The algorithm's steps are as follows:

1. Begin with a small number of manually selected seed words. These seeds form the initial semantic lexicon.
2. Look up members of the lexicon in the text corpus and generate extraction patterns from the contexts of their occurrences.
3. Rank patterns based on a number of factors that reflect their quality. Good patterns are those that are expected to co-occur with other valid category members which are not yet discovered.

4. Apply the best *N* patterns (*Top Patterns*) to the text corpus and record their extractions as candidate lexicon members.
5. Score candidate extractions according to collective evidence from *all* patterns that extract them and add to the lexicon the top *M* ones which have not already been added (*Top Candidates*).
6. Stop if the lexicon has grown large enough or jump to step 2.

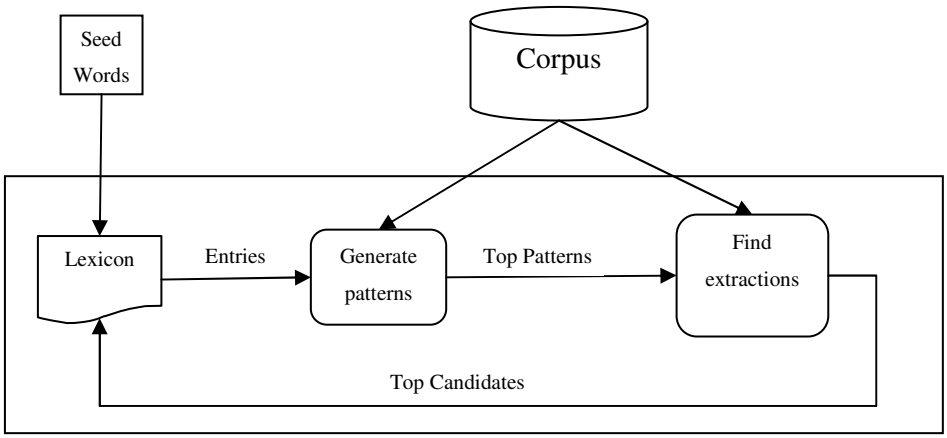


Figure 4: Our bootstrapping process

In the following subsections we explain each step of the bootstrapping algorithm in details:

3.2.1 Extraction Units

Many Question Answering (QA) and Relation Extraction (RE) systems concentrate on broad categories such as PEOPLE or ORGANIZATIONS and use Named-Entity Recognizers (NER) to detect candidate NEs (e.g., (Agichtein and Gravano 2000) and (Banko and Etzioni 2008)). NERs are not applicable to our task for the following two reasons:

1. We would like to build lexicons of *fine-grained* semantic categories whose members are not only proper names (PNs) but also general names (GNs).
2. NERs rely on capitalization to detect PNs. Capitalization guidelines are generally followed only in editorial texts. Unavailable or inconsistent capitalization in various domains (e.g., customer reviews of restaurants and businesses) can mislead NER. Furthermore, in some languages, case does not signal PN at all (e.g., Chinese or German).

Given the above reasons, we must consider a much broader space of candidate names for our task. Most previous works consider NPs (e.g., "Chinese noodles") as candidate lexicon members (e.g., (Riloff and Jones 1999), (Phillips and Riloff 2002), and (Pantel, Crestan, et al. 2009)). Some others consider single nouns (e.g., "pasta") as candidates and stop short of extracting NPs (e.g., (Riloff and Shepherd 1997), (Widdows and Dorow 2002), and (Roark and Charniak 1998)). (Thelen and Riloff 2002) extract NPs in their bootstrapping method but only add the head nouns to the semantic lexicon arguing that the head is the main concept of the NP and is more likely to belong to the semantic category. (Roark and Charniak 1998) suggest a method to include compound nouns in the lexicon in which the head noun is a category member.

English noun phrases (NP) are grammatical units which, in their simplest form, consist of a head noun optionally modified (premodified) by an arbitrary number of modifiers preceding it. Modifiers add information about the head noun and are either determiners (***the*** *salad*), nouns (*the* ***chicken*** *salad*), or adjectives (*the* ***delicious*** *salad*). See Table 8:

Example	Determiner	Adjective	Noun	Head
a delicious pepperoni pizza	a	delicious	pepperoni	pizza

Table 8: A sample noun phrase and its elements

Considering NPs as extraction units is a more realistic approach since semantic classes may include not only single nouns, but also multi-word units (MWUs). However extracting NPs which are valid category members is less straightforward than single nouns:

a. *NP identification*: NP-chunkers have limited performance. Also their reliance on cues such as capitalization and POS tags further decreases their accuracy. Other approaches such as those based on suffix arrays to detect MWUs (Tsai and Chou 2011) extract many spurious collocations unless accompanied by strong heuristics and cues, therefore are not very accurate.

b. *Nested nature of NPs*: Simple English noun phrases are nested; meaning every subsequence of the words in a NP which contains the head noun is itself a valid NP. Here we refer to such subsequences as *NP-suffixes* (Eq. 9). Modifiers provide information about different aspects of the head noun from quality to size, shape, age, color, etc. Depending on the domain in hand, some simply provide extra information (e.g., the *opinion* modifier "good") while others complement the head (e.g., the *color* modifier in "red wine") or sometimes change its meaning. The former can be omitted from the NP without affecting the comprehension of the head while the latter cannot. This

means that not all NP-suffixes might be legitimate category members and methods should be applied to detect valid NP-suffixes from invalid ones. This is, however, a non-trivial domain-specific task and that is why almost all previous works use the NPs output by NP-chunkers without any modifications.

NP	NP-suffixes	valid category member?
a delicious French bread	a delicious French bread	✗
	delicious French bread	✗
	French bread	✓
	bread	✓

Table 9: Nested structure of a noun phrase

In this work, we choose NPs as the extraction units as well. Many open source NP-chunkers are available in the public domain that are accurate and efficient. Also, we experiment different methods for detecting valid NP-suffixes as well as using single nouns as the extraction units and constructing NPs from them accordingly (described in section 4.4).

3.2.2 Seed Selection

Similar to most semi-supervised approaches, we start with a small number of manually selected category members. Some previous works show the effect of seed set composition and size on performance ((Paşca and Durme 2008) and (Pantel, Crestan, et al. 2009)). (Pantel, Crestan, et al. 2009) show that 5 to 20 seeds to start with is optimal and increasing the number of seeds beyond 20 or 30 neither helps finding any new entities, nor decreases the error rate. (Roark and Charniak 1998) emphasize the importance of choosing the most frequent words as seeds in tasks that suffer from sparse data.

On the contrary to (Welty, et al. 2010) who claim that infrequent extraction patterns can only be captured by a large set of seeds, we believe the same effect can be achieved by choosing a small number of high frequency seeds that provide a broad coverage of the category instances. Furthermore, obtaining a large list of trustworthy seeds which truly represent the semantic class is a tedious work and sometimes impossible in domains where knowledge bases may not be existing or rich enough. Given the above, we list the most common nouns in the corpus and manually scan through them and pick the seeds which form the initial lexicon.

3.2.3 Extraction Patterns

Given a NP chunked corpus and the initial lexicon, the second step of our bootstrapping algorithm is to look up the lexicon entries among the NPs in the entire corpus and generate extraction patterns from the contexts of the instances. As for our patterns representation, we choose a window of w_l tokens to the left and w_r tokens to the right of a candidate NP. This patterns representation is computationally cheap and easy to obtain and needs neither parsing of the text, nor any information about syntactic roles of words (Figure 5).

$$Pattern_i = t_{w_l} \dots t_1 \_ t_1 \dots t_{w_r}$$

Figure 5: Window of words as extraction pattern

For every occurrence of a seed, we capture a window of three tokens before and after it. The seed is replaced with a wildcard and all the shorter sub-windows which contain the wildcard are recorded as patterns (see Figure 6). We retain punctuations in the patterns as they can help capture interesting occurrences of the seeds such as in lists (e.g., “turkey, ham, and chicken”). We further filter patterns by retaining only those which are longer than a minimum length.

sample seed occurrence	patterns	length
ordered a large **pizza** with dipping sauce	ordered a large __ with dipping sauce	7
	a large __ with dipping sauce ordered a large __ with dipping	6
	ordered a large __ with a large __ with dipping large __ with dipping sauce	5
	ordered a large __ a large __ with large __ with dipping __ with dipping sauce	4
	a large __ large __ with __ with dipping	3
	large __ __ with	2

Figure 6: Patterns generated from a seed occurrence

Each pattern, when applied to the corpus, extracts NPs, some of which are already known category members (each pattern extracts at least the category member that resulted in its generation). The next step is to score these patterns based on their potential to extract category members. We score each pattern using the well-known $RlogF$ metric used by (E. Riloff 1996) for extraction pattern learning. The $RlogF$ score for each pattern is computed as:

$$RlogF(pattern_i) = \frac{F_i}{N_i} \times log_2(F_i) \quad \text{Eq. 19}$$

Where F_i is the number of distinct category members extracted by $pattern_i$ and N_i is the total number of distinct NPs extracted by the pattern.

We believe a good pattern must have three characteristics: be specific to the category, represent the category well by extracting a diverse set of category members, and have a proper makeup. We describe below how these three characteristics are taken into account in our approach:

1. *Specificity:* a pattern should receive a high score if a high percentage of its extractions are known category members (the $\frac{F_i}{N_i}$ ratio in the $RlogF$ formula).

2. *Category diversity:* besides having high specificity, a good pattern should extract a large number of category members. The $log_2(F_i)$ factor in the $RlogF$ metric boosts the score of such patterns. In other words, it enables a pattern which has a reasonable *specificity* but extracts many category members to score high too. The log is used to smooth the effect of frequency factor F_i for very general patterns that extract a lot of category members as well as spurious NPs. We impose further restriction by keeping only those patterns that extract at least a minimum number of category members ($Min_{CMemebr}$).

3. *Makeup:* punctuations, by themselves, provide little or no information about semantic classes of words. To avoid having patterns that are entirely made up of punctuations, patterns are required to have at least one alphanumeric character. Furthermore, patterns with a length (including the wildcard) less than Min_{length} are discarded to avoid having patterns that are very general or incapable of capturing semantic information about the words they co-occur with.

We presume patterns having the highest $RlogF$ scores are associated with the semantic category the most and are more likely to produce new category members. Therefore, we rank the patterns according to their $RlogF$ scores and select the top N ones (*Top Patterns*). Only extractions by these patterns are considered for addition to the lexicon (*Candidate Extractions*). We observe two types of patterns appear in the patterns ranked list:

a. Patterns whose entire extractions already exist in the lexicon tend to score very high in the list simply because their $\frac{F_i}{N_i}$ ratio has the highest value of 1. These patterns, however, are not useful for expanding the lexicon since they do not extract any new category members. We call them *exhausted patterns* and remove them from the ranked list.

b. Every occurrence of a lexicon entry results in generation of a handful of patterns with some varying only slightly from one another. In our dataset, each NP generates approximately 8 valid patterns on average. It is likely that patterns which are only slightly different, have very similar or even identical extraction sets (e.g., “we ordered a __ with chicken wings” vs. “ordered a __ with chicken wings”). These patterns obtain very close or even the same scores and populate nearby positions in the ranked list. If they happen to score high, they substantially limit the range of extractions to be

considered for addition to the lexicon. To address this issue, starting from the top pattern, we record the extractions and if a pattern does not provide any new extractions we discard it from the ranked list. We call such a pattern a *redundant pattern*.

(Thelen and Riloff 2002) observe as the bootstrapping progresses, the same patterns keep reappearing near the top of the ranked list. This phenomenon happens because only extractions by the Top Patterns are considered for addition to the lexicon and those which find their way to the lexicon boost the $RlogF$ scores of patterns that extract them helping the patterns reappear in Top Patterns. In the absence of competition, only extractions of these patterns will end up being added to the lexicon which degrades the bootstrapping performance. They suggest introducing more candidates in the competition by incrementing the number of Top Patterns by 1 after each iteration. We will put this idea into test in our experiments.

3.2.4 Selecting Lexicon Entries

After selecting the Top Patterns, it is now time to score the Candidate Extractions and add the best ones to the semantic lexicon. Early bootstrapping algorithms such as Meta-Bootstrapping (Riloff and Jones 1999) considered only the extractions of one best pattern for addition to the lexicon whereas more recent bootstrapping methods (e.g., (Thelen and Riloff 2002)) consider a larger set of extractions produced by the top N performing patterns. Subsequently they score the extractions based on collective information gathered from *all* patterns that extract them: an extraction receives a high score if it is extracted by patterns that tend to extract valid category members.

To score the extractions, we take the same approach; however, we adapt a different formula similar to the one used in Snowball for RE (Agichtein and Gravano 2000). Intuitively, the $RlogF$ metric is a weighted conditional probability and can be looked at as the probability of a pattern extracting valid category members. To treat $RlogF$ scores as probability scores, we normalize scores of all patterns in the patterns ranked list by dividing them by the highest score of all. Given an extraction E and the set of patterns $P = \{P_i\}$ that extract it, under assumption of independence of these probabilities, the probability of E being a valid category member $Prob(E)$ can be calculated by the following formula:

$$Prob(E) = 1 - \prod_{i=0}^{|P|}\big(1 - Prob(P_i)\big) \qquad \text{Eq. 20}$$

Where $Prob(P_i)$ (normalized $RlogF$) is the probability of P_i extracting valid category members. Every pattern P_i is believed to extract a number of valid and invalid category members. One can think of $Prob(P_i)$ as the probability of an extraction of P_i being among the valid ones. In this way, $Prob(E)$ can be interpreted as the probability of E being among valid extractions of at least one of the patterns that extract it, calculated by subtracting the probability of its complement (E not being among valid extractions of any of the patterns that extract it) from 1.

Using this metric, all Candidate Extractions are scored and the top five of them are added to the semantic lexicon. The next iteration of the bootstrapping starts over with an expanded lexicon.

3.2.5 Alternative Scoring Methods

Our bootstrapping approach borrows some ideas from Basilisk (Thelen and Riloff 2002) such as $RlogF$ metric for scoring the patterns, limiting the Candidate Extractions to those of Top Patterns, and using collective information from all patterns for scoring extractions. Despite its similarities, it has major differences with Basilisk specifically in scoring the Candidate Extractions. To score a candidate E, Basilisk computes the average logarithm of the number of distinct category members extracted by patterns that extract E:

$$AvgLog(E) = \frac{\sum_{i=1}^{|P|} log_2(F_i)}{|P|} \quad \text{Eq. 21}$$

Where $P = \{P_i\}$ is the set of patterns that extract E and F_i is number of category members extracted by P_i. Hence, a candidate receives a high score if it is extracted by patterns that tend to extract known category members. The logarithm is used to prevent the average from being skewed by one pattern that extracts a large number of category members. We believe this formula has two limitations which are not present in our probability formula:

The first limitation is that the only information this metric uses from a pattern is the number of known category members it extracts (F_i) and ignores pattern's specificity. General patterns might extract a good number of category members, but they also extract many spurious ones. Using only F_i causes invalid extractions of general patterns to achieve higher scores too. Suppose E is extracted by P_1 and P_2 and E' by P_1' and P_2'; and we have $F_1 = F_2 = 3$, $N_1 = N_2 = 10$, $F'_1 = F'_2 = 3$, and $N'_1 = N'_2 = 1000$ respectively. Surprisingly, according to the above formula, $AvgLog(E) = AvgLog(E') = 2$ while

patterns extracting E clearly have higher confidence and therefore E should be receiving a higher score which it is not.

Intuitively, we are not interested in information provided by general patterns for scoring the Candidate Extractions, in the same way we are not interested in including their extractions to this set. By using the normalized $RlogF$ metric, we are practically incorporating patterns' *specificity* and *diversity* into our score and prevent contribution of general patterns.

The second limitation in Eq. 21 is using an Average function. While the arithmetic mean incorporates knowledge of all patterns that extract a candidate, it is not sensitive to the number of patterns. Using that formula, a candidate extracted by one pattern with $F_i = x$, is as likely to be a category member as a candidate extracted by 100 patterns with $F_i = x$.

We believe both number of patterns and their quality are important factors in determining whether a candidate is a true category member or not. Our formula favors candidates extracted by more patterns or more reliable patterns. For example if E is extracted by three patterns of equal probabilities of 0.6 and E′ is extracted by only one pattern with 0.8 probability of extracting valid category members, $P(E) = 0.936 > P(E') = 0.8$ because there is more evidence for E to be a valid lexicon entry although each are individually weaker than the evidence for membership of E'.

3.3 Preventing Drift in the Bootstrapping

3.3.1 Motivation

One problem with bootstrapping methods is that the lexicon being constructed is very vulnerable to drift from the target semantic category. As opposed to Context Vector models where all the candidates get ranked based on their similarity to a fixed set of seeds, in bootstrapping methods, it is the current members of the lexicon that decide what to be added next. When spurious or ambiguous entries are admitted to the lexicon, semantic neighbors of these entries are also likely to be admitted and so on. (Roark and Charniak 1998) call these out-of-category entries "infections". Presence of infection in the lexicon in early iterations, when the lexicon is small and they have more significance, leads to admittance of many more in later iterations in a reinforcing manner and can result in a considerably drifted final lexicon.

Figure 8 visualizes the task of semantic lexicon induction in a hypothetical text corpus (the rectangle). The target semantic category occupies a certain subset of the search space (area with dashed line) and a number of seed entries (the area with solid black background) are provided. As you can see, infections have caused the final lexicon (shaded area) to grow way beyond the target category significantly degrading its quality.

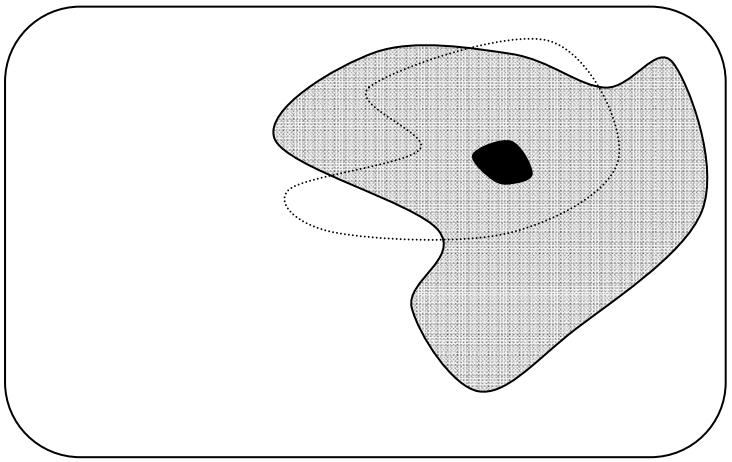


Figure 7: drifted final lexicon

One solution could be to have a human reviewer reject misclassified lexicon entries as the final phase of lexicon construction. However, this is a tedious task and learner does not benefit from it during the bootstrapping process. Another solution is to stop infections from making their way to the lexicon in the first place by, for instance, rejecting the false entries after every iteration. This, especially if done in early iterations, can significantly improve bootstrapping performance. Rejected entries can also be used as some sort of feedback to the learner so it can distinguish ambiguous/low-quality patterns that lead to extraction of infections. Since one of our goals is to eliminate human intervention, in the following subsections, we propose automatic ways to fight infections.

3.3.2 Learning multiple semantic lexicons

When building fine-grained semantic lexicons in specialized domains, one encounters a number of closely related semantic classes with their members co-occurring and sometimes used in the very similar contexts in text. For example in the Medicine domain, Diseases and Symptoms can be easily interchanged in "suffers from __" or in the Terrorism domain, Governments and Terrorist Organizations may both appear in contexts such as "__ bombed". Assuming every word has a single sense per domain (referred to as

"single-sense-per-domain" assumption in (Thelen and Riloff 2002)), we almost always come across confusion errors where a word gets labeled as category *X* while it really belongs to category *Y*. Figure 8 depicts the presence of non-overlapping (non-hierarchical) semantic categories of various sizes in the corpus.

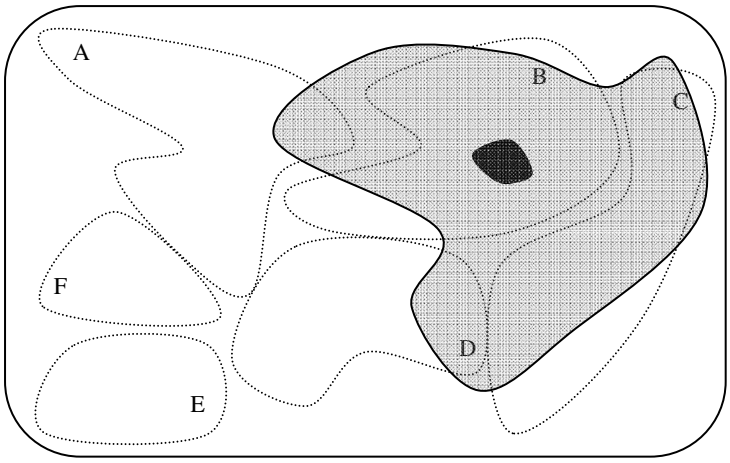


Figure 8: Presence of multiple semantic categories

You can see that a lot of the infections added to the target lexicon actually belong to these semantically related categories. As a solution, one might think of making the learner aware of the existence of multiple categories by building several lexicons simultaneously. As depicted in Figure 9, multiple non-overlapping lexicons being constructed at the same time can restrain each other from occupying territories of one another and help stay within their own boundaries.

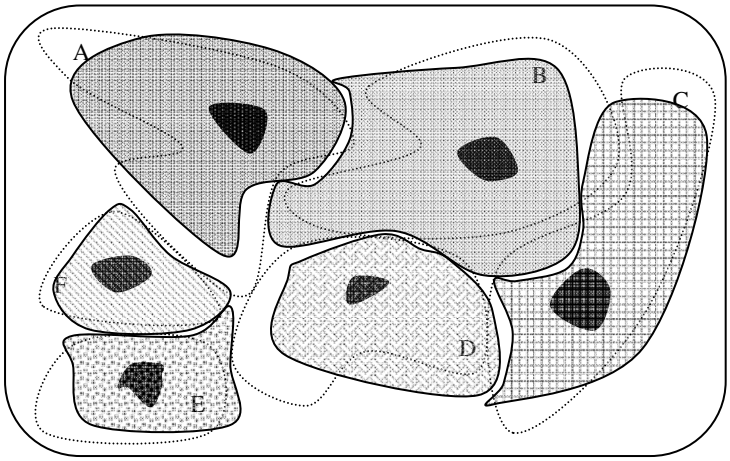


Figure 9: Multi-category lexicon construction

In a multiple category learner, we start with a separate seed set for each category; learning of extraction patterns and selection of candidate extractions are performed individually for every category; and finally sets of non-overlapping entries are added to the lexicons after every iteration. At the end of an iteration, the same candidate might be claimed by multiple competing categories. The most straightforward way of resolving confusion errors is to assign it to the category for which it receives the highest score, provided that it has not been added to another lexicon in the previous iterations.

3.3.2.1 Conflict resolution in Basilisk

(Thelen and Riloff 2002) suggest a more sophisticated conflict resolution method. After ranked lists of candidate extractions are produced for every category, they apply a second scoring function for every candidate-category pair which prefers words that have strong evidence for one category and little or no evidence for the rest. Every candidate extraction E receives a score for category C_i based on the following formula (Thelen and Riloff 2002):

$$diff(E, C_i) = AvgLog(E, C_i) - \max_{j \neq i} \left(AvgLog\left(E, C_j\right) \right) \quad \text{Eq. 22}$$

Where $AvgLog$ is the scoring metric used in Basilisk (Eq. 21). $AvgLog(E, C_x)$ will be zero if E is not claimed by C_x. After $diff(E, C_i)$ is calculated for all candidate-category pairs, the simple conflict resolution idea is used to assign the candidates to their respective categories.

Suppose we have two candidates E_1 and E_2 and they are both claimed by two categories C_1 and C_2. E_1 receives $AvgLog$ scores of 0.9 and 0.8 and E_2 receives 0.8 and 0.1 for C_1 and C_2 respectively. Although there is a stronger evidence on membership of E_2 in C_1 than E_1, a simple conflict resolution picks E_1. The above formula however rescores them to $diff(\mathrm{E}_2, C_1) = 0.7 > diff(\mathrm{E}_1, \mathrm{C}_1) = 0.1$ and picks E_2 which is a more certain choice.

3.3.2.2 Our Approach to Multi-Category Lexicon Construction

Basilisk's multi-category scoring function proves effective in resolving conflicts and assigning the candidates to the correct categories (Thelen and Riloff 2002), however, it does not prevent such conflicts from happening in the first place. Recall that disputed candidates exist because members of some related semantic categories tend to occur in similar contexts. Ambiguous contexts result in the generation of

ambiguous patterns that may perform well in extracting members of the target category but extract members of competing categories too. It is beneficial to punish such patterns so that we have more reliable candidate extractions and fewer conflicts when it comes to assigning them to the categories. Detecting ambiguous patterns is also helpful since, as opposed to Basilisk, we intend to learn effective extraction patterns while constructing a semantic lexicon.

In the presence of multiple competing categories, we can now simulate a human reviewer giving feedback to the learner, which uses this feedback to learn patterns that are more specific to the categories. We take an approach similar to (Agichtein and Gravano 2000) and (Lin, Yangarber and Grishman 2003), and modify the $RlogF$ score by taking into account positive and negative extractions. For each learner, we record the following when applying a pattern (P_i) to the text:

1. Positive extractions ($pos(P_i)$): already existing in the lexicon of the same category.
2. Negative extractions ($neg(P_i)$): already existing in the lexicons of competing categories.
3. Unknown extractions ($unk(P_i)$): not existing in any lexicon yet.

Note that in $RlogF(P_i) = \frac{F_i}{N_i} \times \log_2(F_i)$, $N_i = |pos(P_i)| + |neg(P_i)| + |unk(P_i)|$ and $F_i = |pos(P_i)|$. We discard the patterns with less positive extractions than negative, modify the $RlogF$ metric by replacing F_i with $|pos(P_i)| - |neg(P_i)|$, and rank the patterns accordingly:

$$RlogF_{feedback}(P_i) = \frac{|pos(P_i)| - |neg(P_i)|}{|pos(P_i)| + |neg(P_i)| + |unk(P_i)|} \times log_2(|pos(P_i)| - |neg(P_i)|) \qquad \text{Eq. 23}$$

Keep in mind that the above functions are category specific and the argument C_j is dropped for brevity. The $RlogF_{feedback}$ scores are then normalized ($Prob(P_i, C_j)$) and used in $Prob(E, C_j) = 1 - \prod_{i=0}^{|P|}\left(1 - Prob(P_i, C_j)\right)$ in the same way as before.

By subtracting $|neg(P_i)|$ from $|pos(P_i)|$ in the numerator and the logarithm, the $RlogF_{feedback}$ promotes patterns that have more positive and less negative extractions, thereby punishing ambiguous ones. Although, this is believed to help reduce conflicts, we still need to be ready to appropriately resolve them should they arise. We resolve conflicts in the same way as Basilisk: we rescore the candidates and then use the simple conflict resolution on the new scores. However we use a different formula, which is in fact the extension of our idea to look at candidates' $RlogF_{feedback}$ as probability scores:

Recall that, we consider $RlogF_{feedback}$ score of candidate E for a specific category C_j, the probability of E being a valid member of C_j ($Prob(E, C_j)$). In presence of multiple competing categories, this can be extended in the following way: the probability of E being a valid member of C_j is $Prob(E, C_j)$ times the probability of E not being a valid member of any other category:

$$Prob_{multiC}(E, C_j) = Prob(E, C_j) \times \prod_{k \neq j} (1 - Prob(E, C_k)) \quad \text{Eq. 24}$$

Back to our example of two candidates and two categories, if $Prob(E_1, C_1) = 0.9$, $Prob(E_1, C_2) = 0.8$, $Prob(E_2, C_1) = 0.8$, and $Prob(E_2, C_2) = 0.1$, $Prob_{multiC}(E_1, C_1)$ will be 0.18 and $Prob_{multiC}(E_2, C_1)$ will be 0.72. Besides it being a more natural extension of candidates' probability scores, $Prob_{multiC}(E, C_j)$ metric uses the collective information of *all* competing categories to resolve conflicts as opposed to Basilisk's $diff(E, C_i)$ that looks at only one of the competing categories for which the candidate scores the most. For example, assume we have five competing categories $\{C_1, C_2, C_3, C_4, C_5\}$; a candidate claimed by only C_1 and C_2 with scores {0.9,0.7,0.0,0.0,0.0} is more likely to actually belong to C_1 than when it is claimed by all five categories with scores {0.9,0.7,0.7,0.7,0.7}.

3.3.3 Giving more trust to seeds and early extractions

No matter how well we prevent drift in the lexicon, there will still be some infections added to the lexicon at different iterations. If seed words are selected carefully, we are certain that none of them are spurious or ambiguous. However, we do not have the same level of certainty about the 5 candidates added to the lexicon after the first iteration and since we might already have infections in the lexicon, it is more likely to have them in the next 5 candidates to be added to the lexicon too. As the bootstrapping progresses, we slowly become less and less certain about the validity of entries added to lexicon.

In all the bootstrapping methods proposed in the literature, all existing members of the lexicon equally take part in deciding the best patterns and consequently candidates. It is intuitive to trust the seeds or lexicon entries added in earlier iterations more than those added in the 100th iteration for learning the best patterns. We believe the right level of conservativeness in trusting the new lexicon entries can prevent drift in the learning process and also not harm extraction of new candidates. We propose a degree of trust for lexicon entries and a way to incorporate this into the pattern scoring metric. We represent the degree

of trust to the lexicon entries by a value (between 0 and 1) which is a function of the iteration in which they were added to the lexicon:

$$trust(E) = f(iteration) \quad \text{Eq. 25}$$

In $f(iteration)$, seeds (assumed to be added at iteration 0) must have the maximum value of *trust*, while it steadily approaches zero for those entries added to the lexicon in later iterations. Since the precision in early iterations is normally still very high, to well resemble the reality, an appropriate trust function for our task must provide a slow discount rate in early iterations and maybe a faster rate in later ones. We use the following discount functions in our experiments:

$f(x) = 1 - \left(\frac{x}{maxItr}\right)$
$g(x) = log_{maxItr+1}(maxItr + 1 - x)$
$h(x) = 0.95^x$

Table 10: Different discount functions

Where $maxItr$ is the iteration at which bootstrapping is to be stopped and x is the current iteration. Figure 10 visualizes the functions in Table 10. $f(x)$ is a linear function with a fixed discount rate; $g(x)$ is a logarithmic function which starts with a slow discount rate and gets stricter close to the end; and the third one, $h(x)$, is exponential with a sharp drop in the beginning.

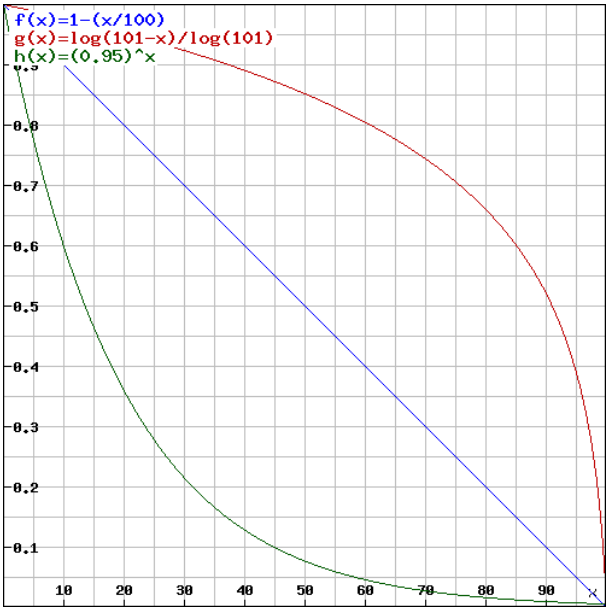


Figure 10: Visual comparison of various discount functions

Now instead of having all the entries participate equally in the $RlogF_{feedback}(P_i)$ metric, each contributes proportionally to its degree of trust. Note that this idea applies to negative extractions in the same way it does to positive ones. An extraction being added to a competing lexicon in early iterations is more likely to actually belong to it than when it is added to it in late iterations. Therefore, we replace $|pos(P_i)|$ and $|neg(P_i)|$ in the numerator of the fraction and the logarithm in $RlogF_{feedback}(P_i)$ metric (Eq. 23) with the following two summations respectively:

$$|pos(P_i)| = \sum_{E \in pos(P_i)} \big(trust(E)\big)$$

$$|neg(P_i)| = \sum_{E \in neg(P_i)} \big(trust(E)\big)$$

Eq. 26

Chapter 4

Implementation and Experiments Setup

4.1 Dataset

Our dataset is a corpus of 157,865 customer reviews of restaurants from CityGrid[5]. The reviews are in English and for 32,782 restaurants in North America. The average number of reviews for a restaurant is 4.8 with a maximum of 327 and minimum of 1. Each review contains information regarding the business such as restaurant ID, restaurant name, review ID, author, review text, rating, pros, cons, and more. For the task of building a semantic lexicon, we are only interested in the review texts, thus, we store them in separate files for each review. Afterwards, we prepare the dataset in the following steps:

1. Convert text to Unicode.
2. Replace URI and HTML encodings with their character equivalent.
3. Replace some special characters of Latin or Germanic languages with their two-character equivalents (e.g., ß → ss or ñ → ny). In all other cases remove the accents.
4. Standardize the spacing between words and punctuations.
5. Replace unambiguous English contracted forms (e.g., “I'm” → “I am” or “you’ll” → “you will”).

Steps 1-4 are performed since, first, the reviews come from an online source[5] and sometimes contain HTML and URL encoded characters, and second, there are discrepancies among authors of the reviews in the use of spacing, capitalizations, or non-ASCII characters in non-English words. The last step however

[5] www.citygrid.com

is optional and merely performed for generalization purposes since our dataset is not very large and thus suffers sparsity.

We continue the preprocessing by passing the corpus through a sentence splitter, a tokenizer, a POS tagger, and a phrase chunker. For this purpose, we use Apache OpenNLP[6], a free NLP toolkit that supports most common NLP tasks including the above. The outputs of OpenNLP tools are compatible with each other and thus their UNIX commands can easily be piped to one another, enabling us to perform the above tasks as simply as issuing a single command:

```
>$ ./opennlp SentenceDetector models/en-sent.bin < inputFile | ./opennlp TokenizerME models/en-token.bin | ./opennlp POSTagger models/en-pos-maxent.bin | ./opennlp ChunkerME models/en-chunker.bin > outputFile
```

It can take up to a several seconds for each of these tools to load. This makes processing single review files inefficient. Therefore, we batch every 1000 reviews into one file and restore them to original after the processing is finished. After this step, we record all NPs in the corpus (over 2 Million) along with information such as the review ID, sentence number, sentence offset, and context in which they appear in (window of 3 words to the left and to the right of the NP) into a database[7]. This way, instead of scanning the entire corpus to find instances of a NP or pattern, we can efficiently use FULLTEXT queries in the database.

4.2 Evaluation set

We would like to evaluate our bootstrapping method for building a lexicon of dish names on our corpus of restaurant reviews. To create an evaluation set, we randomly selected 600 restaurant reviews (approximately 3500 sentences) and had two annotators (annotators A and B) manually label 300 reviews each. In order to have a richer test set which can be used for evaluation of other NLP tasks (e.g., sentiment analysis), annotators labeled a more comprehensive list of things:

[6] incubator.apache.org/opennlp/
[7] We used MySQL.

1. *Dish name*: nouns, NPs, or more complex structures referring to a food (e.g., "seafood", "pepperoni pizza" or "white rice with black beans")
2. *Aspect*: any noun or NP referring to an aspect of a restaurant, such as "server", "waiter", "price", "atmosphere", "service", and "interior".
3. *Positive (Negative) modifier*: a word, phrase, or clause that expresses a positive (negative) sentiment of the author towards a *food* or *aspect* (e.g., "great" in "the sub was great", "enjoyed" in "we enjoyed the food", "one of my favorites" in "Godiva chocolate cake is one of my favorites", or "hate" in "I hate the waitresses there").
4. *Association between a positive (negative) modifier and the corresponding food (aspect)*: for example, to specify "great" refers to "sub" in "the sub was great".
5. *General Positive (Negative) statement*: expressions of opinion that do not fall into the modifier category (e.g., "could use a bit of upgrading" or "highly recommended")

To perform the annotations, we modified a version of Stanford's Annotation Tool[8], a simple open source tool for annotating spans of text with desired labels. Minor modifications were done to enable the annotators specify the association between foods/aspects and modifiers. After labeling was done by annotators A and B, a third annotator (C) went through all labeled dish names and aspects and resolved errors and inconsistencies. The final result was a set of 1007 dish names and 433 aspects.

4.2.1 Training

Before starting the annotation, in order to train the annotators, they were asked to independently annotate the same set of 50 randomly chosen documents. Annotations (for dish names and aspects) by A and B can be visualized in the following contingency table (Figure 11).

[8] http://nlp.stanford.edu/software/stanford-manual-annotation-tool-2004-05-16.tar.gz

		Annotator A: Food	Annotator A: Aspect	Annotator A: None
Annotator B	Food	Both detected As food	A detected as aspect and B detected as Food	B detected as food, unannotated by A
	Aspect	B detected as aspect and A detected as food	Both detected As aspect	A detected as food, unannotated by B
	None	A detected as food, unannotated by B	A detected as aspect, unannotated by B	Don't know how many but we consider it 0

Figure 11: Annotators contingency table

Since annotators were not labeling a fixed set of items (meaning one might label something the other misses), we could not use Kappa[9] statistics to calculate inter-annotator agreement (IAA). Instead we employed the following metric used by (Wiebe, Wilson and Cardie 2005) to calculate the agreement of each annotator with the other:

$$agr(A||B) = \frac{|A\ agreeing\ with\ B|}{|A|} \qquad \text{Eq. 27}$$

As (Wiebe, Wilson and Cardie 2005) point out, "The $agr(A||B)$ metric corresponds to the recall if A is the gold standard and B the system, and to precision, if B is the gold standard and A the system". Therefore, we can use the F-measure (the harmonic mean of precision and recall) as the measure of mutual agreement between A and B. The IAA results are presented below:

	$\boldsymbol{agr(A\|\|B)}$	$\boldsymbol{agr(B\|\|A)}$	**F-score**
Dish Names	0.67	0.73	0.7
Aspects	0.39	0.43	0.41

Table 11: IAA results

Table 11 shows that human annotators can identify dish names with reasonable accuracy. We attribute the lower agreement in detecting the aspects to obscurity of them and high subjectivity of deciding whether or not something is an aspect of a business. After training, the annotation guidelines were

[9] http://en.wikipedia.org/wiki/Cohen's_kappa

modified for more clarity and the annotators were briefed about their errors and inconsistencies. Although not tested, we believe this must have increased the agreement in the actual annotation.

4.2.2 Evaluation set size

Although there is no best size for the evaluation set to have the most reliable results, the rule of thumb is that the evaluation set should be large enough to well represent the corpus. In our preliminary evaluations, we observed a considerable number of valid lexicon entries to be judged negatively because they did not exist in our evaluation set. This caused some runs to seem to perform more poorly than how they actually did.

To make the evaluation set better reflect the corpus, one solution is to make the evaluation set larger. However, it was not possible for us to do this due to lack of resources to annotate more reviews. Instead we did the following: If an NP appears in the 600 reviews that make up our evaluation set, we know it has been observed by our annotators and labeled appropriately; otherwise, we cannot say if it is a valid lexicon member or not. To make sure all the NPs that could possibly be extracted in a run are judged, we retain only those NPs in the corpus that appear in our evaluation set, reducing the number of eligible NP instances to about 1.2 Million. This step can be easily performed with one query in our database.

4.3 Efficient NP and Pattern lookup

The bootstrapping methods described in the previous chapters, either do not provide any details how they were implemented or propose crude and inefficient implementations. The most straightforward way to implement our bootstrapping algorithm would consist of the following steps:

1. Lookup instances of lexicon entries in text and generate extraction patterns.
2. Search for these patterns in text, record their extractions and assign them a score based on the extractions. Good patterns give us our Candidate Extractions set.
3. Lookup Candidate Extractions in text and record all the patterns that co-occur with them.
4. Again, search for these patterns in text and assign them a score based on their extractions. Use these scores to select future lexicon entries.

5. Repeat all previous steps now with a larger lexicon.

Performing all these lookups take tremendous amounts of time if the corpus is to be scanned. Each lexicon entry (especially seeds which we choose among the most frequent words in the corpus) co-occur with a great number of patterns. This means, in every iteration, hundreds of thousands of patterns are looked up in the corpus and this number increases as the lexicon grows. Although efficient, FULLTEXT search is not fast enough for this volume of queries.

To achieve a $O(1)$ lookup time, before the bootstrapping, we generate extraction patterns for every NP in our corpus in an exhaustive way. For each unique NP we record all the unique patterns with which it co-occurs and the frequencies of co-occurrence, and for each unique pattern we record all the unique NPs it extracts and in how many occasions. For example:

a) The noun phrase “Pizza” co-occurs with “__ was great” N times, “pineapples on the __” M times, etc.

b) The pattern “two slices of __” co-occurs with “pizza” X times, “cake” Y times, etc.

Given the above data, we store two matrices (NP-Patterns and Pattern-NPs) on the disk, replacing patterns and NPs with unique IDs. Since these matrices are sparse, they can be loaded into memory using efficient data structures that allow the fastest lookup time and the least memory usage. We use hash maps for this purpose and load the matrixes into two separate *Python Dictionaries*, one having the NPs as keys and lists of Patterns as values and the other having Patterns as keys and lists of NPs as values. We further increase the speed by reducing the number of pattern lookups that need to be done in every iteration by caching scores of the patterns to use in the next iterations. Only scores of those patterns that co-occur with the newly added seeds have to be recalculated.

4.4 Extraction Units

As mentioned previously, semantic classes of nominals may include single nouns as well as Multi-Word Units (MWU). For this reason, we choose to consider noun phrases for inclusion in our semantic lexicon. One could propose two possible approaches to this:

1. Use NPs as units of extraction.
2. Extract single nouns and then rank NPs in the corpus according to these nouns.

Either of the two approaches has its own justification. For instance, extracting NPs is more straightforward because an NP could be a MWU or a single noun, so one does not need to deal with complications of ranking NPs based on the single nouns they contain. On the other hand, single nouns are more frequent than NPs and therefore more context is available to do the bootstrapping. Less sparsity helps the algorithm better distinguish valid category members from invalid ones resulting in better performance.

4.4.1 Extracting Noun Phrase

English NPs, in the simplest form, are made up of a head noun (the rightmost noun in the NP) and a number of modifiers preceding it. These modifiers can be determiners (e.g., articles or quantifiers), adjectives, and nouns. NP-chunkers tend to group all theses modifiers modifying the head noun into one single unit, which (excluding determiners) conforms to the following regular expression:

$$(Adj)^*(Noun)^*Noun$$

We restrict our candidate NPs to only those having this form. These NPs, however, may or may not be suitable for addition to the lexicon as output by the NP-chunker. Here we propose and evaluate a variety of methods to identify NP-suffixes that represent a legitimate dish name:

4.4.2 Removing the Subjective Modifiers

In addition to determiners, NPs may contain modifiers that are not part of the multi-word term. In our domain, noun modifiers and those adjective modifiers that describe color, shape, age, or place of origin are highly likely to be part of the multi-word term that refers to a food. Examples are: “chicken pasta”, “red wine”, “square pizza”, “old-fashioned hamburger”, or “Italian pastrami”. However, quantity and quality/opinion (subjective) modifiers generally are not part of dish names (e.g., “two Kabobs” or “appetizing salad”). Removing these modifiers is a crucial step in the task of term recognition from opinionated data (e.g., customer reviews), where subjective modifiers are abundant.

The most straightforward way to tackle this problem is to detect and remove these modifiers. Determiners and quantity modifiers are accurately detected by POS taggers and are removed from all candidate NPs beforehand. Provided that we are given a lexicon of subjective adjectives, we can detect and remove them as well in the following two steps:

1. In each NP, find the rightmost occurrence of a subjective adjective.
2. Remove all words preceding (and including) this adjective in the NP.

The rationale behind this method is that, in English, the premodifying adjectives usually occur in a specific order and those that precede a subjective adjective are normally not part of the multi-word term. The order of adjectives from left to the right is more or less: opinion, appearance (size, shape, length, or condition), age, color, origin (nationality or religion), and material (Hogue 2003). A modifier that precedes a subjective adjective can only be an adverb, another subjective adjective, a number, or a determiner.

In product reviews domain, (Hu and Liu 2004) report that customers tend to use similar words when commenting on product features. Same conclusion could be drawn for restaurant reviews and in fact can be observed in the manually labeled data. Thus, the lexicon of subjective adjectives we use does not need to be very large and therefore is not very difficult to obtain. We use a lexicon of 1505 subjective adjectives collected using the semi-supervised approach introduced in (Vechtomova and Robertson 2011) to extract and rank all the adjectives in the corpus by similarity to a small set of seed adjectives. Top 2000 extractions of different runs were examined and those found to be subjective formed the lexicon.

(Riloff, Wiebe and Wilson 2003) successfully use Meta-Bootstrapping and Basilisk for the extraction of subjective *nouns* (e.g., love, hate, or trust). Our pattern representation and bootstrapping framework are general enough that there is no reason it cannot be used to build a semantic lexicon of subjective *adjectives*. In fact, the only thing that needs to change is the extraction units. To achieve full automation, one can simply take the output lexicon at a desired size and treat its members as subjective adjectives. However, with no human intervention, precision of a lexicon of adequate size will be obviously not so high. We perform experiments to evaluate the performance of our bootstrapping algorithm in learning subjective adjectives.

4.4.3 Include NP Suffixes in the Candidate NPs

As mentioned before, we assume noun modifiers are part of the multi-word terms (provided that the POS tagger detects them correctly), while some adjective modifiers may not. The fact is, these modifiers are not limited to subjective or quantity adjectives; depending on the dish name, adjectives of age, shape, etc. also might not be part of the multi-word term.

A powerful bootstrapping algorithm must be capable of detecting a small number of valid lexicon entries among thousands of invalid candidates; and it does so using the words' contexts. In fact, according to the distributional similarity hypothesis, the context of a word can be used to determine its semantic class. As a result, contexts of occurrences of phrases such as "Italian pizza" and "pasta" should reflect their membership in the *dish names* category but contexts of "delicious Italian pizza" should not. For example, "Italian pizza" is likely to be preceded by words such as "delicious", "thin-crust", and "tasty" throughout the text. Presence of such modifiers in contexts of a word is a strong indication that the word in question is a food; on the other hand "delicious Italian pizza" is very unlikely to appear with any of those modifiers or in general any subjective modifier.

Based on this fact, we hypothesize that if we augment the list of candidate NPs with NP-suffixes generated by removing adjective modifiers one at a time, the bootstrapping process will be able to select those that are valid category members and discard the rest based on their context. For example, every occurrence of "delicious Italian pizza" gives artificial occurrences of "pizza" and "Italian pizza" as well. As shown in Table 12, each NP-suffix has a different context (three tokens to the left and to the right of the NP-suffix) and extraction patterns for it are generated the same way as before.

Original NP	Sub-phrases and their context
enjoyed the ***delicious Italian pizza*** on whole grain	enjoyed the ***delicious Italian pizza*** on whole grain
	enjoyed the delicious ***Italian pizza*** on whole grain
	the delicious Italian ***pizza*** on whole grain

Table 12: NP sub-phrases and their context

The strengths of this approach are its simplicity, no use of external resources, and the fact that more than one valid suffix might be selected for addition to the lexicon (in Table 12, both "Italian pizza" and "pizza" are valid lexicon entries). A drawback of it might be the increase in the size of NPs and extraction patterns to consider (in our dataset the number of NPs increased by approximately 1.3 times). However, the increased volume can be handled by our efficient implementation.

4.4.4 Detecting the Most Stable MWUs

C-value is one of the well-known, but not widely used, methods for multi-word term recognition (Frantzi and Ananiadou, Extracting nested collocations 1996). C-value is a domain independent statistical method

used for extraction of *nested terms* by scoring them according to their stability ("termhood") in the corpus. *Nested terms* are those that appear within other longer terms and may or may not appear by themselves in the corpus. The measure is based on statistical characteristics of the candidate term and is defined as follows:

$$C-value(a) = \begin{cases} log_2|a|.f(a) & a\ is\ not\ nested \\ log_2|a|\left(f(a) - \frac{\sum_{b \in T_a} f(b)}{|T_a|}\right) & otherwise \end{cases} \quad \text{Eq. 28}$$

Where a is the candidate term, $|a|$ is the length of a in terms of words, $f(.)$ is the frequency of non-nested occurrence in the corpus, and T_a is the set of candidates that contain a. C-value discounts the score of a candidate by subtracting the number of times it appears within other longer terms, unless it appears in many of them which is a sign of its independence. It also takes into account the length of the term because the corpus frequency of a longer term is more significant than that of a shorter one. For more details on the rationale behind this score see (Frantzi, Ananiadou and Mima 2000).

If one is to find out whether a modifier is part of the dish name before the bootstrapping starts and without using external resources, there seems to be only one clue, and that is corpus statistics reflecting the termhood of NPs. Therefore, we use the C-value metric in the following way aiming to detect the most stable multi-word term in a nested NP:

1. For every candidate NP, generate all its NP-suffixes (including itself) with the shortest being all the nouns in the NP.
2. Calculate the C-value for all the suffixes and replace the highest scoring one with the original NP.

Figure 12 shows the NP-suffixes and their C-value scores for a sample NP, where NN indicates a noun and JJ an adjective.

$$\frac{lukewarm}{JJ}\ \frac{creamy}{JJ}\ \frac{tomato}{NN}\ \frac{sauce}{NN} \rightarrow \begin{cases} \frac{tomato}{NN}\ \frac{sauce}{NN} & 266.99 \\ \frac{creamy}{JJ}\ \frac{tomato}{NN}\ \frac{sauce}{NN} & 6 \\ \frac{lukewarm}{JJ}\ \frac{creamy}{JJ}\ \frac{tomato}{NN}\ \frac{sauce}{NN} & 2.32 \end{cases}$$

Figure 12: All nouns are present in NP-suffixes for which C-value is calculated

This method, however, has disadvantages such as the fact that there is no specific stability threshold that enables us to select multiple stable NP-suffixes in a NP. Also *frequency* and *length* may not be sufficient indications to determine *termhood* in general.

4.4.5 Extracting Single Nouns

In this approach, we bootstrap single nouns that belong to the semantic category, and then rank the candidate NPs according to the nouns they contain. Our bootstrapping algorithm can be easily adapted to extract single nouns, by omitting the NP chunking phase and considering any token tagged as *Noun* by the POS tagger as a candidate entry. The rest of the algorithm (including the definition of patterns) remains the same. To rank the NPs, we need to come up with a degree of category membership for single nouns first:

Although, in every iteration, candidate extractions are ranked based on some scores, these scores cannot be used to rank the lexicon entries globally. However, we can have a global ranking in which an entry E_1 is ranked higher than E_2, if it was added to the lexicon in an earlier iteration, or had a higher rank than E_2 in the iteration in which both were added to the lexicon. Having a ranked list of lexicon entries, we can now assign them a score based on the following formula:

$$score(E) = \frac{1}{rank(E) + C} \qquad \text{Eq. 29}$$

Where $rank(E)$ is the rank of lexicon entry E and C is a smoothing constant.

For the NP scoring function, we propose a simple sum of the scores of the nouns contained in a NP. One should note that while head nouns almost all the time belong to the category themselves, non-head nouns may or may not be legitimate category members. Thus, we rank only those NPs whose head noun exists in the lexicon of single nouns.

Intuitively, the head noun of an NP represents the main concept of the NP and further away a noun is from the head, the less it does so. Therefore, it makes sense to reduce contribution of the nouns to the overall score of the NP according to their distance from the head of the NP (Eq. 30):

$$NPScore = \sum_i \left((1 - log_{10}(d_i)) \times w_i\right) \qquad \text{Eq. 30}$$

Where w_i is the score of the noun i and d_i is its distance (in tokens) from the right end of the NP (d of the head is 1). For example, if the score for "pizza" in our lexicon of dish names is 0.8 and that of "trucker" is 0.1, "trucker pizza" receives a higher score than "pizza trucker" since pizza's 0.8 will be discounted in the latter case.

4.5 Dependency Triples as Context

We want to investigate whether semantic classes of words can be determined solely by the words around them in absence of more constraining linguistic information such as their syntactic roles in sentences. Although knowing syntactic roles of words can help better distinguish their semantics, parsing large corpora of text to obtain such information is expensive, imprecise, and unscalable. We particularly compare using grammatical relations of words in sentences as context against the window of words surrounding them. For this purpose we use grammatical dependency relations employed by several previous works (e.g., (D. Lin 98) or (Vechtomova and Robertson 2011)). We perform the following steps to obtain these dependency relations:

1. After POS tagging the corpus, we parse it using the Stanford Dependency Parser[10] (De Marneffe, MacCartney and Manning 2006). Output dependency relations are in the form of triples which consist of two words, their POS tags, and the dependency relation between them (e.g., "eat VB:dobj:NN pasta").
2. For each noun in the corpus, we record the dependency triples in which it participates and transform them into extraction patterns by replacing the noun in question with a wildcard "__". For example if the noun in question is "pasta", the triple <eat, VB:dobj:NN, pasta> will be transformed to <eat, VB:dobj:NN, __>.
3. For each unique noun, record the unique patterns that extract it along with the frequency of co-occurrence. Do the same for patterns and construct the matrices Noun-Patterns and Pattern-Nouns accordingly.

[10] Any other dependency parser such as Minipar (D. Lin 1993) can be used as well.

One should note that dependency grammars are different from phrase structure grammars (constituency grammars) in the sense that they lack phrasal nodes; meaning that dependency relations are between single words and not phrases (verbal phrases, noun phrases, etc.). In fact, the structure in the dependency grammar is determined by the relations between heads of phrases and their dependents. The left side of Figure 13 shows the output generated by the parser for the sentence "I/PRP had/VBD a/DT delicious/JJ chicken/NN salad/NN"; on the right side, you see visualization of these relations:

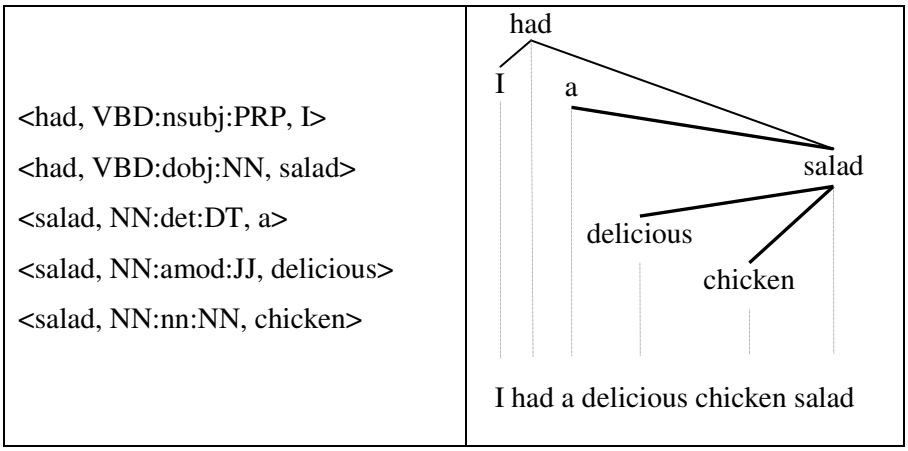


Figure 13: Example of dependency grammar triples

Dependency grammar parsers are capable of detecting phrase boundaries although they are not shown in the dependency relations. By looking at the relations in which the head and the modifiers of "a delicious chicken pasta" participate, you can see that the modifiers only participate in relations with the head (shown in bold lines) and only the head ("salad") participates in relations with other words in the sentence. In order to use dependency triples as context for NPs, we need to modify them in a way that they reflect relations between phrases.

Given the candidate NPs, one can ignore the dependency relations within NP boundaries and replace the head of the NP with the entire NP in every dependency relation outside of the NP boundaries. We call this process *collapsing* and perform it for candidate NPs whose head participates in dependency relations. Although *collapsing* can be done for all phrase types (e.g., verbal or adjectival), we only perform it for NPs. We believe collapsing other phrase types makes the already sparse set of tuples even sparser. Table 13 shows the dependency triples and their graphical representation after the only candidate NP in this sentence ("chicken salad") is collapsed:

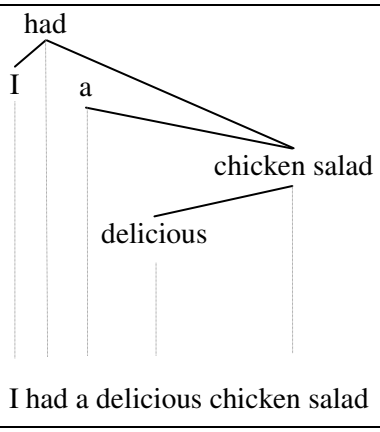
<had, VBD:nsubj:PRP, I> <had, VBD:dobj:NN, chicken salad> < chicken salad, NN:det:DT, a> <chicken salad, NN:amod:JJ, delicious>	

Table 13: Dependency triples after collapsing "chicken salad"

Note that collapsing depends on the NP-chunker and the parser to agree in detecting NP boundaries to work effectively. We believe this agreement is high since both the NP-chunker and the parser take the same POS tagged text as input[11].

4.6 Experiments Setup

In this section we outline evaluation goals and describe how the eperiments are performed. In the next two chapters we will present the evaluation results, discussions, conclusions, and possibilities for future work.

4.6.1 Seed Selection and Parameter Tuning

To evaluate extraction of dish names, we ctreate two sets of seed words each containing 10 seeds (Table 14). Set 1 is used for tuning the parameters and set 2 for testing purposes. These sets are created by listing the most common words in the corpus and manually selecting the seeds into the sets alternating between the two.

[11] Future versions of Stanford Parser will be accepting text with annotated phrase boundaries.

Set 1	Set 2
chicken	pizza
sauce	sushi
salad	wine
cheese	steak
fish	bread
soup	meat
beef	seafood
rice	shrimp
coffee	dessert
pasta	fries

Table 14: Sets of seed words used for extraction of dish names

We run our bootstrapping for one hundred iterations which results in a lexicon of five hundred entries. In some of the experiments, along with the lexicon of dish names, we construct a lexicon of *aspects of business*[12]. In the reviews, customers tend to comment not only on foods, but also a wide range of aspects of the restaurants from *value* to *atmosphere*, *service*, *wait time*, *menu variety*, etc. Furthermore, as mentioned before, people have a tendency to use a very similar language to express their opinion about things. This similarity of context causes a lot of aspects to be falsely detected as dish names; thus, we believe learning lexicons of these two simultaneously can increase the learning precision.

Since the *aspects* category is coarse and heterogeneous, we use a larger and more diverse set of seeds (Table 15), aiming to represent aspects of different types. Note that this does not prejudice our evaulation results since our primary goal is to evaulate the precision of learning a lexicon of dish names and not a direct comparison between performance of learning dish names and aspects.

[12] We will refer to them as "aspects" for brevity.

Aspect seed set	
place	area
service	price
restaurant	dishes
menu	waiter
staff	waitress
wait	portions
atmosphere	location
bar	owner
table	manager
prices	selection

Table 15: Set of seed words used for extraction of aspects

As the baseline, we extract noun phrases whose subjective modifiers are removed using our manually judged lexicon of subjective modifiers. Extraction patterns are scored by Eq. 23 and the extractions by the top N of patterns are considered as Candidate Extractions, which are in turn scored by equation Eq. 24. Our algorithm has three parameters which need to be tuned:

1. *The minimum acceptable length for extraction patterns (Min_{length}).* The maximum length is fixed to seven (three tokens on the left, three on the right, and the wildcard in the middle). Longer patterns tend to be very specific and score low in the ranked list of patterns.
2. *The minimum number of category members a pattern needs to extract* ($Min_{Cmember}$).
3. *The number of top scoring patterns which give the Candidate Extractions (N).*

4.6.2 Our Bootstrapping Method vs. Basilisk

To see how our method performs compared to the state of the art bootstrapping algorithms, we evaluate it against Basilisk (Thelen and Riloff 2002) for extraction of dish names in our corpus of restaurant reviews. We use the best performing setup of parameters obtained in the previous experiment to test our method against Basilisk when both learn two lexicons of dish names and aspects at the same time.

We implement Basilisk with the original settings and formulas in (Thelen and Riloff 2002), and use the grammatical dependency triples output by the Stanford Dependency Parser, to generate Basilisk's extraction patterns according to the heuristic rules of AutoSlog (Table 7).

Also, to compare the effectiveness of our formulas and methods for selection of patterns and lexicon entries against those of Basilisk's, we perform another experiment where Basilisk uses our lexical patterns instead of the lexico-syntactic patterns generated by AutoSlog. This tells us how much of the difference in the perofrmances comes from the different types of extraction patterns used and how much comes from the bootstrapping algorithms themselves.

4.6.3 Drift Prevention Methods

To evaluate the effectiveness of techniques proposed in section 3.3 for preventing drift in the bootstrapping, we conduct four runs. In the first run, we construct a lexicon of dish names when it is the sole lexicon being learned; in the second experiment, we also learn a lexicon of aspects at the same time; in the third experiment, we incorporate the negative feedback in our scoring formula (Eq. 23); and finally we test assigning a degree of trust to the lexicon entries. We employ various discount functions (presented in Table 10 and depicted in Figure 10).

Although not tested, it is posible to divide the aspect seeds into more fine-grained homogeneous categories and bootstrap more than two lexicons where we believe our multi-category scoring metric for the extractions (Eq. 23) particularly works better than that of Basilisk (Eq. 22).

4.6.4 Multi-word Term Detection Approaches

In section 4.4, we proposed four approaches to finding the proper NP-suffixes for addition to our lexicon of dish names. These four methods are 1- removing subjective modifiers from NPs, 2- adding NP-suffixes to the list of candidate NPs, 3- using the C-value metric (Frantzi and Ananiadou 1996) to replace the NPs with their most stable NP-suffix, and finally 4- extracting single nouns and rank candidate NPs based on the nouns they contain.

So far, we used NPs with their subjective adjectives removed as the baseline of our algorithm. We would like to know whether the above techniques help detect valid dish names, and also compare their effectiveness. We evaluate methods 1-3 against a run where no modification is done to NPs output by the

NP-chunker. In the fourth method, the metric used to rank the candidate NPs does not take subjective modifiers into account; therefore NPs which only differ in those modifiers receive identical scores. This means that the top ranks in the final ranked list can potentially get populated with entries which are composed of high-scoring NPs being premodified by various subjective modifiers, while none of them are legitimate dish names. To be able to more fairly compare the effect of extracting single-nouns on the bootstrapping, we use the candidate NPs from the first run (with subjective modifiers removed) and compare the fourth run against the first.

4.6.5 Lexical vs. Dependency Triples Patterns

In this experiment, we investigate the effect of using grammatical dependency relations as context on the bootstrapping performance. In order to do so, we evaluate our multi-category bootstrapping when lexical patterns are used against when patterns based on dependency triples (generated as described in 4.5) are employed. Except for the type of the patterns, the bootstrapping details, including the extraction units, is identical in both cases.

4.6.6 Boostrapping vs. Context Vector Models

Another interesting evaluation to perform is to compare bootstrapping methods against context vector models. The former consider words' contexts as patterns that are matched in text and extract members of a semantic category; while the latter use vectors of contexts to measure semantic similarity of words with one another. Context vector models are also less prone to drift due to ranking the candidate category members according to their similarity to a fixed set of seeds; while in the bootstrapping, all members of the lexicon take part in deciding the next entries. It is worthwhile to see how these two perspectives compare and also how much we are able to control the drift using the measures introduced in 3.3.

For the context-vector model, we use the semi-supervised approach by (Vechtomova and Robertson 2011) to extract dish names. It uses dependency triples to form feature vectors for candidate entries and seeds and ranks the candidate entries based on their similarity to the set of seeds (details are given in section 2.1.4). We perform two runs for each model, one using NPs and the other single nouns as extraction units. Furthermore, we use patterns based on dependency triples in our bootstrapping method to eliminate the effects of using lexical patterns on its performance.

4.6.7 Bootstrapping Subjective Adjectives

To detect and remove the subjective adjectives from NPs, we have been using a manually judged lexicon of subjective adjectives. Another possibile way is to adapt our bootstrapping method to extract subjective adjectives and use the output lexicon for this purpose, provided that our lexicon of subjective adjectives is high precision enough. To see whether full automation is feasible, we perform experiments to evaluate the performance of our bootstrapping method in learning subjective adjectives.

We do so by changing the extraction units to single words tagged as adjectives and provide the bootstrapping with a seed set of subjective adjectives. In fact, we create two sets (Table 16) from the most frequent subjective adjectives in the corpus in the same way as the seed sets of dish names (described in section 4.6.1). We tune the parameters on set 1 and use set 2 for evaluation. Our relevant set will be the lexicon of 1505 subjective adjectives used previously.

Set 1	Set 2
good	great
nice	best
better	friendly
delicious	excellent
bad	fresh
wonderful	favorite
tasty	amazing
awesome	perfect
authentic	rude
average	fantastic

Table 16: Sets of seed words used for extraction of subjective adjectives

4.6.8 The Effects of the Number and Composition of the Seeds

In this experiment, we first analyze the effect that the number of seeds has on performance of the bootstrapping. The values we examine for the size of the seed set are 5, 10, 15, and 20. In order to minimize the random effect of individual seeds on performance, we perform 5 runs for each set size value using a different combination of seed words in every run. The seeds in each run are randomly drawn from

a pool of 30 most frequent dish names (which includes those in Table 14). We calculate the Mean Average Precision (MAP) by averaging the Average Precision (AveP) values for each of the 5 runs and use this metric for evaluation.

(Welty, et al. 2010) hypothesize that, in the task of Relation Extraction, the frequency distribution diagram of extraction patterns has a very long tail and a small set of seed tuples stops short of discovering infrequent but maybe useful lexical patterns (they use 20K seed tuples in their method). In order to evaluate this hypothesis, we look into the frequency distribution of extraction patterns in the corpus, identify those patterns that co-occur with the seed words, and see whether infrequent patterns are discovered too and to what degree.

(Roark and Charniak 1998) suggest that the initial seeds should be among the most frequent words in the corpus to provide the broadest coverage of category occurrences from which additional likely category members will be selected. We are particularly interested in seeing whether selecting the most frequent seeds makes up for the small size of our seed set. In order to do so, we build two sets of seed words (Table 17), one from the most frequent dish names in the corpus and the other from less frequent ones (frequencies below 1000). We compare how these two seed sets perform at discovering the infrequent patterns through analyzing the frequency of the patterns they co-occur with (details are given in Section 5.8).

Most frequent seeds		Less frequent seeds	
Dish name	**Freq.**	**Dish name**	**Freq.**
pizza	21441	bacon	950
chicken	13746	ravioli	944
sushi	10432	burritos	934
sauce	9371	scallops	913
wine	8536	mushrooms	869
salad	8262	noodle	766
steak	8247	turkey	742
cheese	7132	sashimi	734
bread	5632	tapas	697
fish	5557	sangria	682
meat	5130	tofu	681
soup	5042	cookies	656
seafood	4956	cheesecake	649
beef	4885	cocktails	645
shrimp	4784	martini	621
rice	4617	eggplant	620
dessert	4441	tempura	535
coffee	4357	gravy	491
fries	4314	mussels	485
pasta	4306	guacamole	471

Table 17: Sets of most frequent and less frequent seed words

Chapter 5

Evaluation Results

In this chapter, we present the results of the experiments outlined in the Chapter 4, also analyze and discuss them. To evaluate our semi-supervised bootstrapping method for extraction and ranking of multi-word units that belong to a fine-grained semantic class, we use it to extract dish names from a corpus of 157,865 restaurant reviews. We run the bootstrapping for 100 iterations so that 500 lexicon entries are extracted (5 per iteration). Our relevant set consists of dish names labeled in 600 randomly chosen documents.

One of the measures we use for evaluation is Average Precision (AveP). AveP is defined as the average of the precision values after each relevant dish name is encountered in the ranked lexicon entries. It is in fact the area under the Precision-Recall curve and is calculated according to the following formula:

$$AveP = \frac{\sum_{k=1}^{n}\big(P(k) \times rel(k)\big)}{|relevant\ set|} \qquad \text{Eq. 31}$$

Where n is the size of the lexicon and $rel(k)$ equals 1 if the item at rank k is a valid dish name and 0 otherwise. We use AveP as a single-valued measure for comparison across different runs and not as a measure of the overall performance of a run individually. This is because we retrieve only 500 lexicon entries and the size of the relevant set is more than twice as this number (1007 entries), therefore, our reported $AveP$ is lower than that of a system that ranks all the candidate entries (e.g., context vector models).

Another measure used is Precision at specified cutoff values of 50, 100, 250, 350, and 500, denoted as *P@N*. Furthermore, to visually analyze the bootstrapping performance over time, we use a graph whose X axis is the number of lexicon entries learned and Y axis is the number of correct ones.

5.1 Parameter Tuning

Our bootstrapping method has three tuning parameters: minimum acceptable length for extraction patterns (Min_{length}), minimum number of category members a pattern needs to extract ($Min_{Cmember}$), and the number of Top Patterns (N) whose extractions are considered for addition to the lexicon (Candidate Extractions). To tune these parameters, we used our baseline multi-category bootstrapping which learns two lexicons of dish names and aspects using dish names in "Set 1" in Table 14 and aspects in Table 15 as seeds. It uses NPs whose subjective modifiers are removed as the extraction units.

Since Min_{length} and $Min_{Cmember}$ are correlated, we performed 30 runs with different combinations of values these two can take. Min_{length} was evaluated with values 3, 4, and 5 and $Min_{Cmember}$ with values from 1 to 10. Figure 14 shows the AveP for different combinations of the above values.

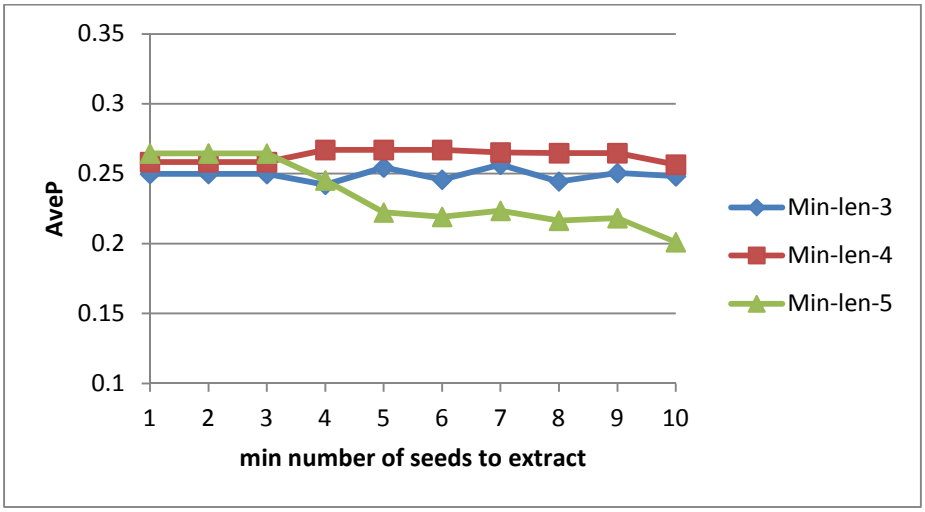


Figure 14: AveP for runs with different values of $Min_{Cmember}$ and Min_{length}

Generally, longer patterns are more specific to the category and give higher precision extractions. This can be observed in Figure 14. Longer patterns are, however, less frequent: In a separate experiment, top 10 best performing patterns of every iteration were collected totaling 362 unique patterns. 47% of the patterns had a length of 4, 41% of the patterns had a length of 3, and only 12% had lengths of 5 or more.

Lower frequency means having fewer category member extractions. This explains the drop in the performance of "Min-len-5" with $Min_{Cmember}$ values of greater than 3, as very few patterns with minimum length of 5 can be found to co-occur with a high number of category members. This drop,

however, is not as sharp as expected since this effect is somewhat alleviated as the bootstrapping continues and extractions of those patterns are added to the lexicon. Also, the reason for the identical performances of runs with $Min_{Cmember}$ values of 3 or less is that patterns which extract fewer than 3 category members never score high enough to take part in the bootstrapping.

Overall, Figure 14 suggests that a minimum length of 4 provides the best trade-off between specificity and category diversity and increasing the $Min_{Cmember}$ up to the mid-range values helps select more accurate patterns. Therefore we set the values of Min_{length} and $Min_{Cmember}$ to 4 and 5 respectively.

To find the best value for N, we tried different values of 10, 15, 20, and 25 for the first iteration (N is incremented by one after each bootstrapping iteration). AveP values obtained for these runs, although not significantly different, suggest a value of $N = 10$ for the first iteration, which allows only those patterns which are strongly associated with the category to be considered in early iterations. Figure 15 depicts the AveP of different runs (the points labeled as "growing"):

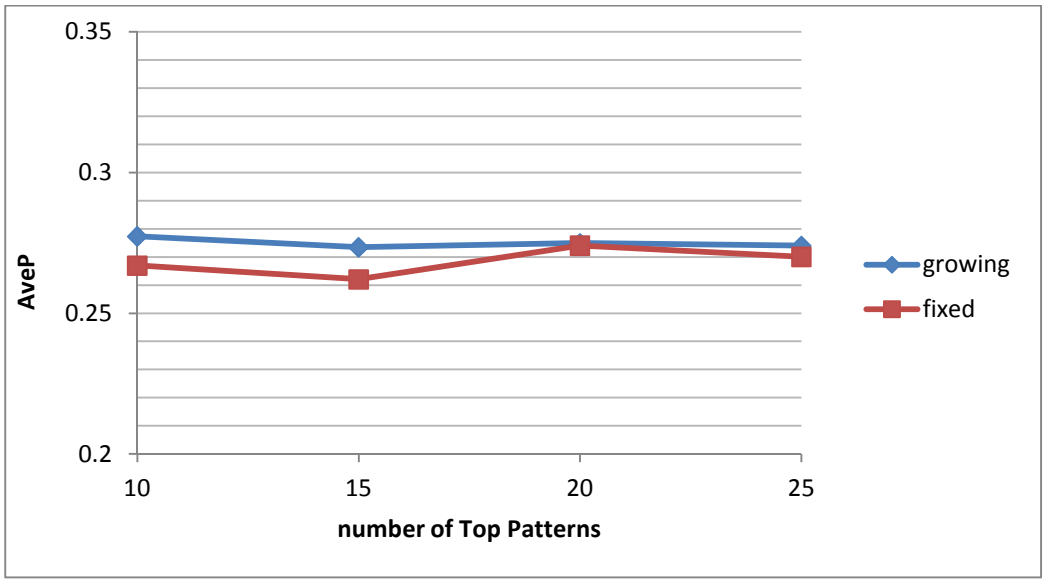


Figure 15: The effect of number of Top Patterns on bootstrapping performance

To verify the observation made in (Thelen and Riloff 2002) about the stagnation of Top Patterns when N is not incremented after each iteration, we performed the same runs as above, this time without incrementing N (points labeled as "fixed" in Figure 15). The results show that having a fixed value for N through the bootstrapping has a slight negative effect on the overall performance. This effect is more considerable when N is small.

5.2 Our Bootstrapping Method vs. Basilisk

To evaluate our bootstrapping method against Basilisk (Thelen and Riloff 2002), we set its tuning parameters to the values obtained in the previous experiment and learn two lexicons of dish names and aspects simutaneously ("Multi-F-Dlog"). We also perform two runs for Basilisk, one using its original extraction patterns ("Basilisk") and the other using our lexical patterns ("Basilisk-lexical"). Basilisk also learns a lexicon of aspects as well as dish names at the same time. The seed words used in this experiment are dish names in "Set 2" (Table 14) and aspects in Table 15. The results are presented in Table 18:

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Multi-F-Dlog	0.272	0.96	0.89	0.827	0.764	0.729	0.68
Basilisk	0.241	0.74	0.75	0.74	0.728	0.703	0.662
Basilisk-lexical	0.247	0.72	0.71	0.72	0.744	0.757	0.666

Table 18: Our bootstrapping method vs. Basilisk

Although "Multi-F-Dlog" extracts only slightly more valid lexicon entries compared to "Basilisk", (note the P@500), our method performs better in early iterations and thus has a higher AveP. This steeper slope can be visually observed in Figure 16 until extraction of nearly 100 entries.

The results in Table 18, also suggest that the lexico-syntactic patterns used in Basilisk do not provide any performance gain compared to our lexical patterns. We believe the reason for the almost identical performance of these two types of patterns is that our lexical patterns, in most cases, implicity comply to the syntactical heuristic rules of AutoSlog. Furthermore, they are capable of capturing occurances of category memebers which do not fit those rules. This can make up for the extra precision that lexico-syntactic patterns may provide.

Therefore, the superior performance of our method compared to Basilisk can be attributed to our stricter scoring formula for selection of lexicon entries (Eq. 20) which, by using the $RlogF$ metric (Eq. 19), takes into account the category specificity of the patterns ($\frac{F_i}{N_i}$) rather than a simple count of valid category members a pattern extracts (F_i). Furthermore, using a $Min_{Cmember}$ threshold retains only those patterns which are highly associated with the category. Such strictness, however, is most effective in the early iterations of the bootstrapping when the size of the lexicon is still small and the lexicon entries are highly likely to be valid category members.

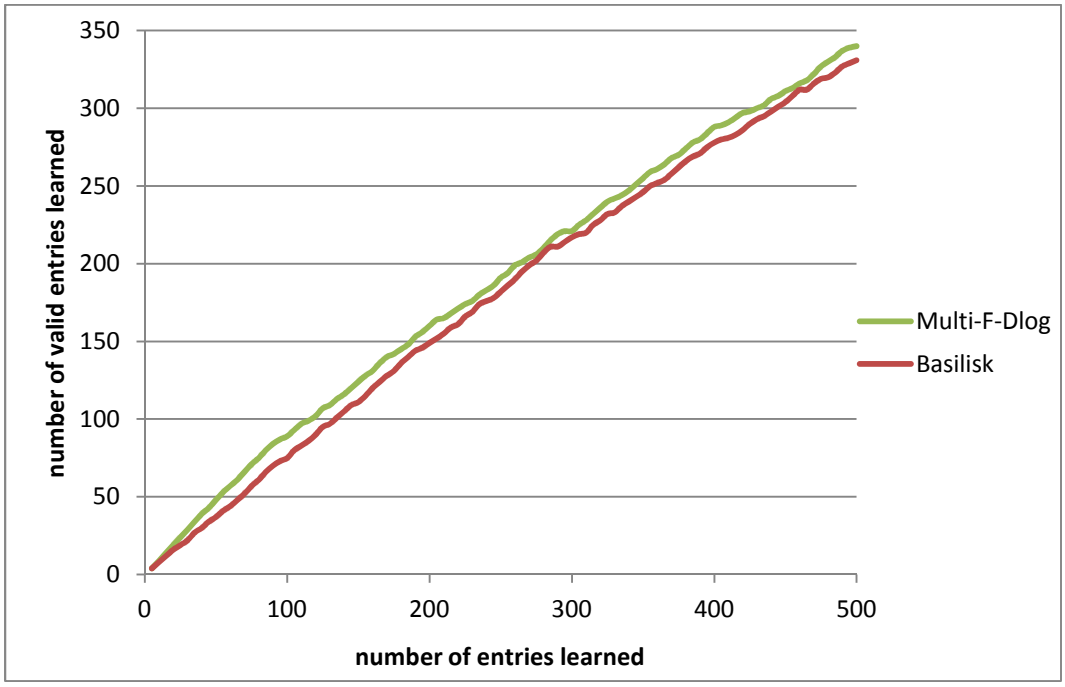


Figure 16: Our bootstrapping method vs. Basilisk

5.3 Drift Prevention Methods

To see how much each component of our method contributes to its overall performance and evaluate the effectiveness of techniques we employ to prevent drift in the bootstrapping, we perform three runs each adding a feature to the previous one. In the first run we learn only a lexicon of dish names in absence of any other competing categories. We call this run "Single". In the second run, a lexicon of aspects is constructed at the same time using the multi-category scoring formula (Eq. 24) for conflict resolution. We identify this run with "Multi". The third run (named "Multi-F") incorporates the negative feedback to the scoring formulas to better guide the bootstrapping. We evaluate these runs against "Multi-F-Dlog" from the previous experiment; it complements "Multi-F" by the use of degrees of trust to the lexicon entries provided by the logarithmic function $g(x)$ in Table 10. The results are presented in Table 19 and Figure 17.

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Single	0.046	0.32	0.35	0.347	0.304	0.3	0.274
Multi	0.214	0.98	0.88	0.8	0.724	0.614	0.542
Multi-F	0.263	0.96	0.89	0.827	0.764	0.726	0.66
Multi-F-Dlog	0.272	0.96	0.89	0.827	0.764	0.729	0.68

Table 19: Performance of different measures for controlling drift in bootstrapping

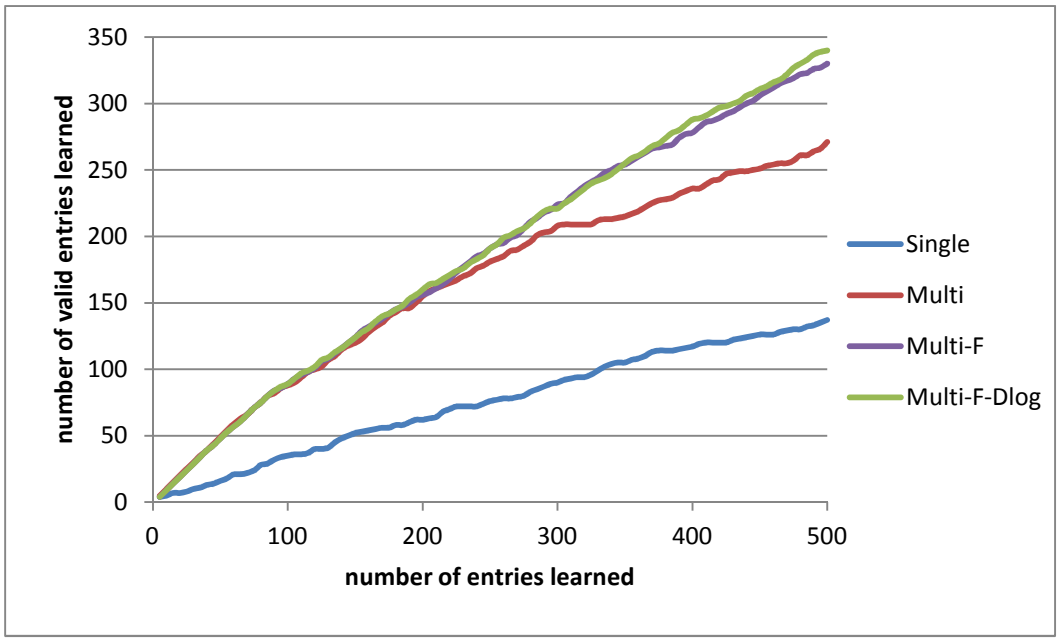


Figure 17: Performance of different measures for controlling drift in bootstrapping

As it can be seen in the results, bootstrapping a lexicon of aspects at the same time as dish names, can significantly improve the learning accuracy. This means many invalid entries arise from confusion errors and learning multiple semantic categories simultaneously can effectively keep the categories within their boundaries. The next idea which considerably improves the performance is to use negative feedback from competing categories to select patterns which are associated with the current category the most.

Results in Table 19 and Figure 17 also show that giving more trust to high precision entries added to the lexicon in early iterations can help control the deviation in final iterations to some extent. The improvement in precision of "Multi-F-Dlog" agrees with the sharp drop in the logarithmic discount function $g(x)$ in Table 10; we expect this improvement to be more noticeable if the bootstrapping was to continue beyond 100 iterations. We also experimented with linear $f(x)$ and the exponential $h(x)$ discount

functions (Table 10) in two runs named “Multi-F-Dlinear” and “Multi-F-Dexp” respectively. the results are presented in Table 20:

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Multi-F-Dlog	0.272	0.96	0.89	0.827	0.764	0.729	0.68
Multi-F-Dlinear	0.259	0.98	0.88	0.82	0.756	0.714	0.654
Multi-F-Dexp	0.240	0.84	0.78	0.76	0.72	0.683	0.648

Table 20: Performance of different discount functions

None of the two other discount functions improves the performance over “Multi-F”, and as expected, the performance deteriorates in “Multi-F-Dexp” compared to “Multi-F-Dlinear” which itself performs worse than “Multi-F-Dlog”. We believe a proper discount function should follow the natural decline in the precision of the lexicon being costructed; more aggressive discount functions tend to do more harm by limiting the scope of the learner.

5.4 Multi-word Term Detection Approaches

In this experiment, we compare the following four runs using “Set 2” (Table 14). The results are presented in Table 21 and Figure 18:

1. Unmodified: NPs as output by the NP-chunker.
2. C-Value: C-value metric is used to replace the NPs with their most stable NP-suffix.
3. NP-Suffixes: NP-suffixes are added to the list of candidate NPs.
4. Subjective-Mod: subjective modifiers are removed from NPs.

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Unmodified	0.229	0.94	0.9	0.8	0.704	0.683	0.598
C-Value	0.222	0.92	0.84	0.78	0.696	0.64	0.604
NP-Suffixes	0.199	0.92	0.85	0.78	0.708	0.6	0.536
Subjective-Mod	0.272	0.96	0.89	0.827	0.764	0.729	0.68

Table 21: Different multi-word term detection approaches

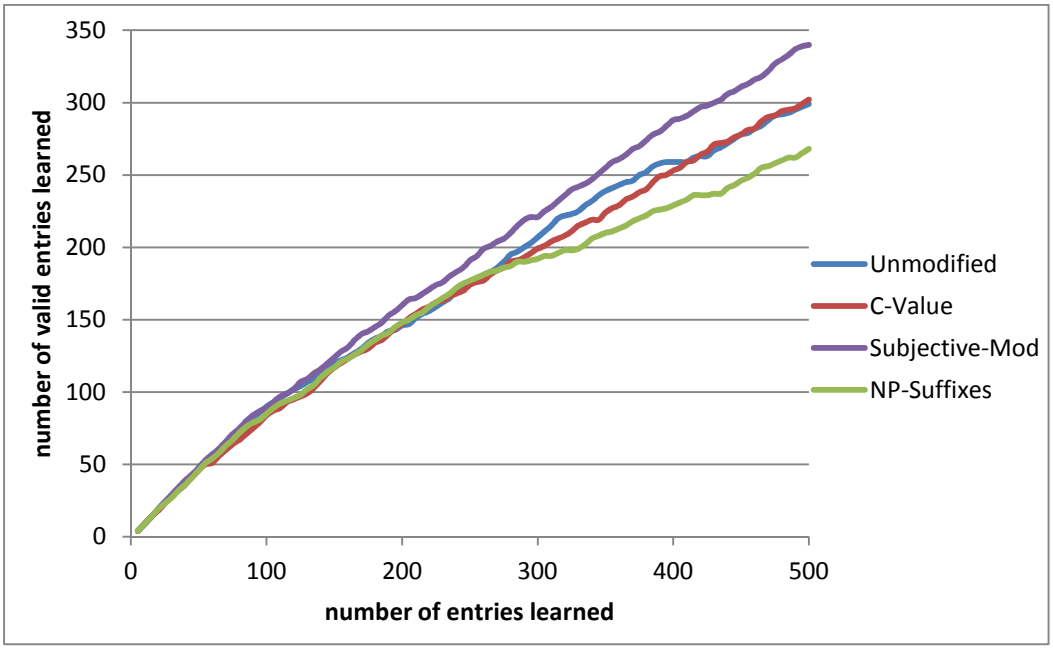


Figure 18: Different multi-word term detection approaches

According to the results, the only approach that helps better identify valid multi-word dish names nested within NPs, is the most straighforward one, which is removing the subjective modifiers and those modifiers appearing before them from the NPs.

The underperformance of C-value in detecting the most stable NP-suffix can be attributed to limitations of corpus statistics in deciding the termhood of a MWU, particularly in sparse data. Another reason might be the bias of C-Value towards shorter terms; this can be observed in the extractions of "C-Value" with the average number of words per extraction being less than the rest of the runs (1.34 against 1.48 in "Unmodified" and 1.42 in "Subjective-Mod" and "NP-Suffixes").

Our hypothesis for the third approach (adding the NP-suffixes to the list of candidate NPs) was that existence of subjective modifiers in contexts of valid NP-suffixes can result in powerful patterns that can extract them with high accuracy (e.g., "they have delicious __"). Looking at the Top Patterns shows that in fact a very small number of them have a subjective modifier appearing immediately before the wildcard; and the most powerful patterns are typically those capturing lists, conjunctions, or other grammatical forms in which dish names appear with high confidence (e.g., "the __ was undercooked").

In another run, we built a lexicon of 500 single nouns and used them to rank the candidate NPs from the run "Subjective-Mod" according to section 584.4.5. Single nouns are less sparse than NPs and we

hope that with more context, the bootstraping performs better in extracting single-word dish names. We tuned the parameters for this run ("Single-Nouns") in the same way as before and found the best results with $Min_{length} = 4$ and $Min_{Cmember} = 4$. We also used $N = 20$ in the first iteration and incremented it by one after every iteration. Table 22 presents the evaluation results using "Set 2" (Table 14):

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Single-Nouns	0.261	0.82	0.84	0.807	0.74	0.72	0.69
Subjective-Mod	0.272	0.96	0.89	0.827	0.764	0.729	0.68

Table 22: Extraction of NPs versus ranking them based on single nouns they contain

The results show that bootstrapping NPs yields slightly better overall performance compared to bootstrapping single nouns and ranking the NPs accordingly. Higher precision of "Subjective-Mod" in early iterations, makes it more suitable for construction of small high-precision lexicons. The reason for the lower precision of "Single-Nouns" at smaller cutoff values can be the scoring metric used for ranking the candidate NPs (Eq. 30). NPs that have the same head nouns are likely to receive similar or identical scores causing invalid entries whose head noun is a valid dish name to appear throughout the final ranked list, regardless of the score of the head.

Since we do not have an evaluation set for single-word dish names, we cannot evaluate the performance of bootstrapping single nouns by itself. Therefore, it is not clear how much of the performance in "Single-Nouns" comes from the bootstrapping and how much from the scoring metric used to rank the NPs. Although here we used only one socring metric, it is possible to try different formula and disocunt factors to have a comparative evaluation of their performance.

5.5 Lexical vs. Dependency Triples Patterns

In this experiment we use our best performing run "Multi-F-Dlog" to compare the performance of lexical patterns against patterns based on grammatical dependency triples. We refer to these runs as "Lexical" and "Dependency" respectively. The "Dependency" run has two tuning paramers: the minimum number of category members a pattern has to extract ($Min_{Cmember}$) and the number of Top Patterns in the first iteration (N). The best results using "Set 1" were obtained with $Min_{Cmember} = 7$ and $N = 20$. The evaulation results using "Set 2" are shown in Table 23 and also visually demonstrated in Figure 19.

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Lexical	0.272	0.96	0.89	0.827	0.764	0.729	0.68
Dependency	0.149	0.7	0.61	0.587	0.524	0.506	0.504

Table 23: Lexical patterns vs. grammatical dependency based patterns

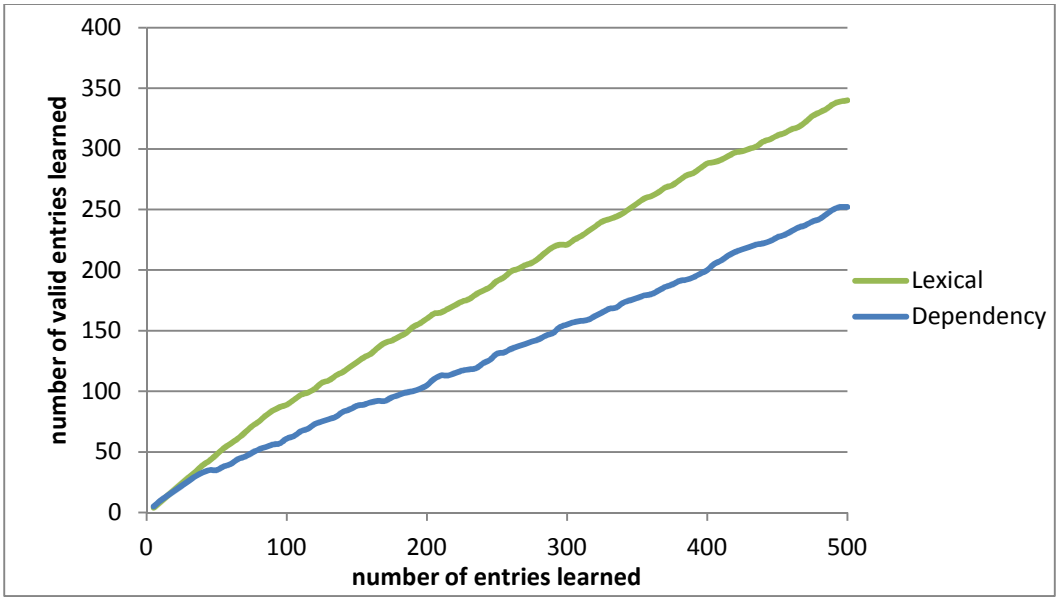


Figure 19: Lexical patterns vs. grammatical dependency based patterns

These results suggest that using simple windows of words as the context is a better choice than grammatical dependencies between words, for extraction of dish names. It seems that despite the extra constraints grammatical dependencies impose on words regarding their syntactic roles, lexical patterns more effectively capture semantic classes of words. We attribute this difference in performance mainly to the following reasons:

The first reason can be the sparsity of grammatical dependencies. In this experiment, the number of lexical patterns with a minimum length of four exceeded 6M while there was only over 800K grammatical dependency patterns available for bootstrapping. Obviously, the fewer the patterns, the lower the chances are for a valid category member to be extracted.

Another reason can be the difference in the amounts of information contained within the two types of patterns. An individual grammatical dependency pattern captures one syntactic relation between a candidate NP and a single word. In other wrods, dependency triples decompose the sentence into single units of information. For example the sentence "I had *beef* on rice" gives the patterns

"had_VBP:dobj:NN_X" and "X_NN: prep_on:NN_rice" for *rice*; while the lexical pattern "I had __ on rice" seems to contain the information of the two dependency patterns in one place. We believe this extra information as well as the order of appearance of the words in lexical patterns tell us more about the semantics of the word in question than knowing its syntactic role in the sentence and its grammatical relations with these words separately.

And last but not least is the ability of lexical patterns to extract instances of category members with high confidence in lists or conjunctions (e.g., "sushi , __ , and maki"). Although dependency parsers detect conjunctions (e.g., "chips_NNS:conj_and:NN_salsa"), lexical patterns can capture high accuracy lists of more than two category members and those that are made up of punctuations such as commas.

5.6 Boostrapping vs. Context Vector Models

In this experiment, we evaluate our bootstrapping method against the conext vector model proposed in (Vechtomova and Robertson 2011). Vechtomova et al. (Vechtomova, Ahmadi and Suleman 2012) use this model on the same dataset of restaurant reviews to rank single nouns according to their similarity to a set of seed dish names. Afterward, they rank NPs (whose subjective modifiers are removed) employing various discount factors, including the one used in this work.

Here we performed the same procedure, only with the difference that we used the top 500 nouns to rank up to 500 NPs. We call this run "ContextVector-Nouns" and compare it against our method where grammatical dependency triples are used to extract the nouns ("Bootstrapping-Nouns"). In both runs, the same set of 1505 subjective adjectives was used to remove the subjective modifers and NPs were ranked according to Eq. 30. We also performed another run ("ContextVector-NPs") using the collapsed dependency triples to directly extract NPs; we compare it against our "dependency" run which we here refer to as "Bootstrapping-NPs".

Using "Set 1", we obtained the best results for "Dependency-Nouns" with $Min_{Cmember} = 3$ and $N = 20$ for the first iteration. The context vector model used in (Vechtomova and Robertson 2011) has three tuning parameters: b and k_1 (Eq. 11), and *seed-threshold*, which is the number of seed words a feature has to co-occur with in order to be included in the feature vectors. The tuning parameter b was evaluated with the values from 0.1 to 0.9 in the incremenets of 0.1, k_1 with the values from 0.2 to 1.6 in the incremenets of 0.2, and *seed-threshold* in the range from 1 to 9. The best results for "ContextVector-

Single-Nouns" and "ContextVector-NPs" were obtained with $b = 0.9$, $k_1 = 1.6$, and 1 for *seed-threshold*. The evaluation results using "Set 2" are presented in Table 24 and Table 25.

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Bootstrapping-Nouns	0.209	0.82	0.76	0.733	0.692	0.62	0.59
ContextVector-Nouns	0.173	0.88	0.75	0.667	0.604	0.566	0.508

Table 24: Bootstrapping vs. Context-Vector Model (single nouns)

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Bootstrapping-NPs	0.149	0.7	0.61	0.587	0.524	0.506	0.504
ContextVector-NPs	0.135	0.9	0.8	0.693	0.54	0.451	0.384

Table 25: Bootstrapping vs. Context-Vector Model (noun phrases)

According to the results, our bootstrapping method, not only has a better AveP in both noun and NP extraction, but also extracts considerably more valid dish names compared to the context-vector model. It is also interesting to note that single noun runs (Table 24) show improvements over NP runs (Table 25) as opposed to when using lexical patterns (section 5.4).

In both lexical and grammatical dependency patterns, sparsity is reduced when extraction units are single nouns; however, we believe grammatical dependency patterns benefit more from it, because the entire extra context (which was lost during the collapsing process) becomes available to head nouns and not nouns modifying the head. This makes extraction of head nouns (which are more likely to be dish names) more accurate and therefore increases the performance of the NPs ranking process.

5.7 Bootstrapping Subjective Adjectives

To tune and evaluate our bootstrapping method for learning a lexicon of subjective adjectives, we use the seeds in "Set 1" and "Set 2" in Table 16 respectively. Our relevant set is the lexicon of 1505 subjective adjectives previously used to remove the subjective modifiers from the candidate NPs.

In the parameter tuning phase, the best performance was obtained with $Min_{length} = 5$, $Min_{Cmember} = 5$, and $N = 10$ for the first iteration. The bootstrapping, however, did not show much sensitivity to the threshold $Min_{Cmember}$. Many patterns could be found that, despite the length of 5, co-occurred with a

large number of category members. This verifies the hypothesis that people tend to use a very similar language to express opinions about things. In other words, subjective modifers tend to occur in very similar contexts, resulting in extraction patterns which are both specific and category diverse. The evaluation results are given in Table 26. This table also presents the results of using the conext vector model of (Vechtomova and Robertson 2011) for the same task with the tuning parameters set to $b = 0.9$, $k_1 = 1.2$, and 2 for *seed-threshold*.

Run	AveP	P@50	P@100	P@150	P@250	P@350	P@500
Bootstrapping-Adj	0.258	0.98	0.99	0.967	0.936	0.877	0.836
Context-Vector-Adj	0.286	1.0	0.99	0.98	0.96	0.95	0.89

Table 26: Extraction of subjective adjectives

The results for "Subjective-Adjectives" show that the performance of our bootstrapping method in extraction of subjective adjectives is clearly superior to that of nouns or noun phrases. We believe this superior performance is due to the highly similar contexts subjective adjectives occur in. A subjective adjective (or an adjective in general) typically comes in two forms[13]:

1. *Attributive Adjective*: is part of the noun phrase headed by the noun it modifies; and generally precedes its noun (e.g., "I love their greasy burger").
2. *Predicative Adjective*: is linked via a copula (be, seem, appear, etc.) to the noun or noun phrase it modifies (e.g., "their burger is greasy").

As it can be seen, there is not much variability in the makeup of contexts (except for the NP whose head is being modified), resulting in precise patterns. Furthermore, these results were obtained while the lexicon of subjective adjectives was the only lexicon being learned. Although not experimented, it might be possible to learn a lexicon of non-subjective modifiers (e.g., adjectives pertaining shape or color or proper adjectives) simultaneously and increase the learning precision.

However, according to Table 26, our bootstrapping method still performs worse than the context vector model which uses grammatical dependency triples. We believe this difference comes from the power of dependency triples in detecting subjective modifiers. In section 5.6, we mentioned three reasons for

[13] There are other types of adjectives (e.g., nominal or absolute adjectives) which are far less common than the two types mentioned here.

inferior performance of grammatical dependency triples in extracting nouns/NPs. In the case of subjective modifiers, however, none of those conditions seem to exist:

Due to the similarity of contexts in which subjective modifiers appear, dependency triples are not sparse anymore. In fact, subjective modifiers mostly appear in only two types of dependencies: <(food/aspect) NN:nsubj:JJ __> which indicates a predicative adjective, or <(food/aspect) NN:amod:JJ __> which stands for an attributive adjective. Furthermore, subjective modifiers normally take part in only one grammatical relation in a sentence and it is with the noun they modify. Therefore, the fact that the dependency triples represent only one syntactic relation should not affect their performance in detecting the subjective modifiers. Subjective modifiers are also not very likely to occur in lists and normally appear in conjunction relations if they are to co-occur with another modifier; this corresponds to the <(adjective) JJ:conj:JJ __> triple.

Since we are able to learn subjective modifiers with a high enough precision, it is worthwhile to experiment using these adjectives to remove the subjective modifiers from NPs, so that our method does not rely on any external resource. We perform an experiment where we remove from the NPs output by the NP-chunker, the top a adjectives in the output lexicon of the "Bootstrapping-Adj" run. We evaluated values of a from 0 to 500 in the increments of 100. Zero here means no adjectives were removed. The tuning parameters were set to the values obtained in section 5.1. The Average Precisions of these runs are demonstrated in Figure 20 and are compared to that of "Multi-F-Dlog" from Table 20:

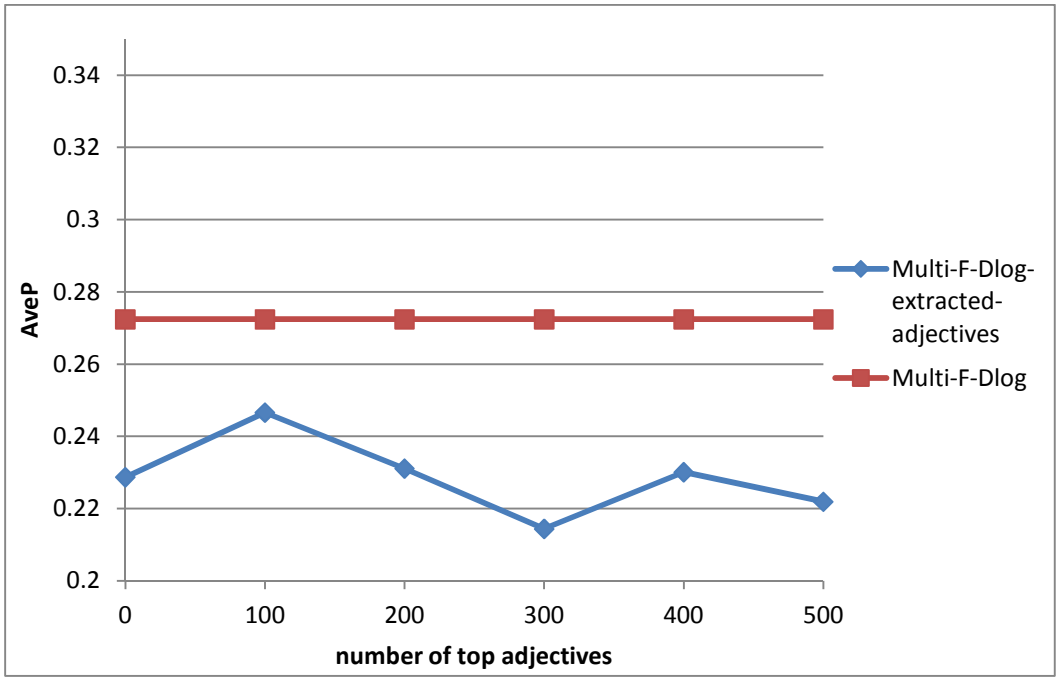


Figure 20: AveP of Multi-F-Dlog using different values of a vs. using the manually judge lexicon

The best results using the automatically constructed lexicon were obtained with $a = 100$ which shows some improvements over when the NPs are unmodified; while "Multi-F-Dlog" still perform the best simply because of using a much larger of set of manually selected subjective modifiers. Using $a = 200$ only very slightly improves upon $a = 0$ and using greater values only decreases the Average Precision. Such behavior is not unexpected because inclusion of non-subjective or ambiguous modifiers (such as roasted, Italian, sour, chopped, etc.) in the lexicon can eliminate a great number of valid dish names from the list of candidate NPs. Nevertheless, these results suggest that removing even a very small but high precision set of subjective adjectives from the candidate NPs, can cause improvements if our method is to be fully automatic.

5.8 The Effects of the Number and Composition of Seeds

Figure 21 shows the Mean Average Precision (MAP) values for different sizes of the seed set (*num-seeds*). As described in section 4.6.8, we evaluated values from 5 to 20 in increments of 5; and for each of these values calculated the MAP on 5 sets randomly drawn from a pool of 30 most frequent dish names.

Standard deviation error bars are also shown to illustrate the effect of randomness in the selection of seeds.

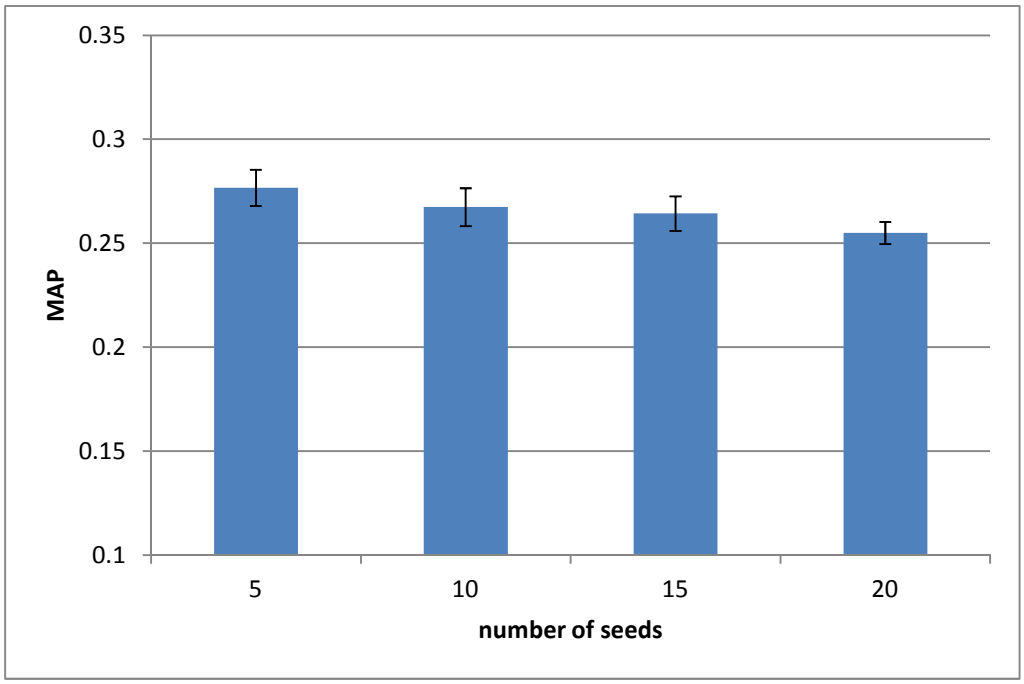


Figure 21: MAP values of runs with different number of seeds

While 5 seeds performed the best, the difference in performance is not large between different values that were tested. There is also a slight difference between 10 and 15. The standard deviation is also small for all the values, which indicates a consistent performance. This value, however, is the lowest for 20 which is due to the larger set size and therefore greater number of common seeds across the 5 runs. Altogether, the results show that increasing the number of seeds, even beyond 5, does not introduce any performance gain.

Now we evaluate the hypothesis by (Welty, et al. 2010) that states frequency distribution of patterns has a long tail and only a large set of seeds can discover infrequent but maybe useful patterns. We do so by analyzing the frequency distribution of extraction patterns in one of our runs, identify those patterns that co-occur with the seed words, and see to what extent infrequent patterns are discovered.

Among over 6.2M patterns, we select 495,290 patterns that co-occur with at least two distinct noun phrases (patterns with only one unique extraction are always discarded in our method). The most frequent pattern occurs 645 times in our dataset. Figure 22 illustrates the frequency distribution for these patterns[14]:

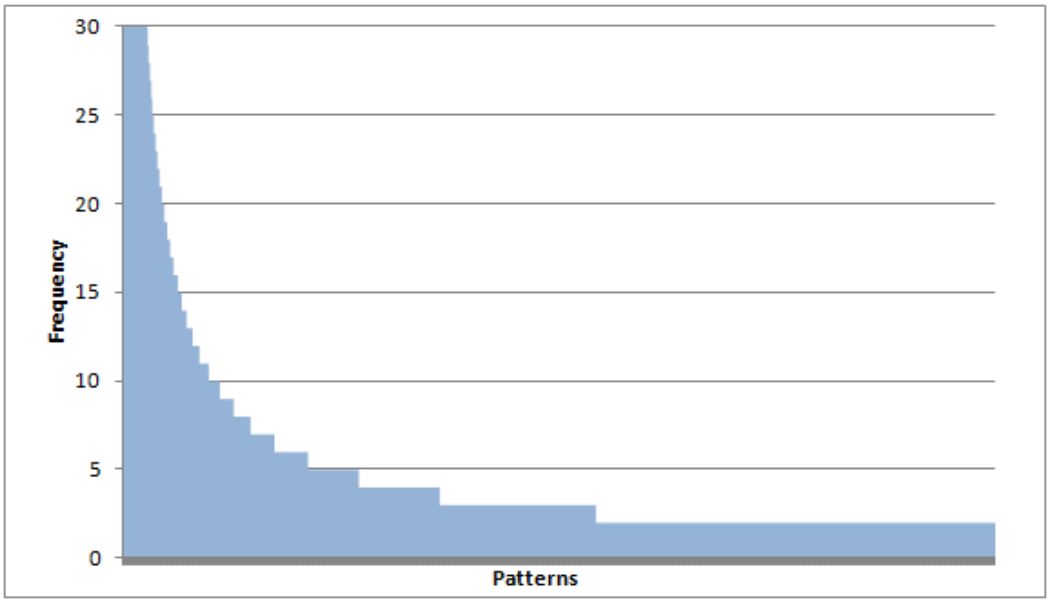


Figure 22: Frequency Distribution of patterns

If we define the *tail* as patterns that appear 3 times or less, Figure 22 well depicts its length. In fact around 63% of the patterns appear 3 times or less. The next step is to identify patterns that co-occur with the sets of most-frequent and less-frequent seeds (Table 17), to see what proportion of the patterns they discover and to what extent they can account for the long tail. The set of most frequent seed words can discover 83,647 patterns from the total of 495,290. Here by *discover* we mean whether a pattern co-occurs with any of the seed words in the set. Frequency distribution of the discovered patterns (darker color lines) is depicted along with the frequency distribution of the rest of the patterns (lighter color lines in the background):

[14] Due to presentation limitations, this diagram demonstrates a 32,000 point sample of this data. Also frequency values are shown up to 30.

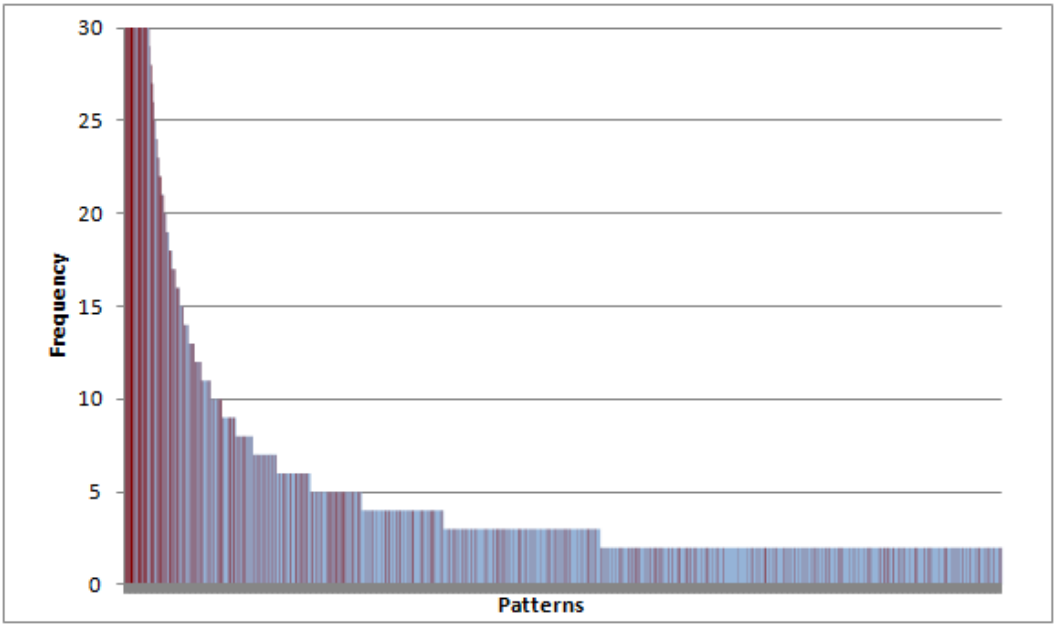


Figure 23: Frequency Distribution of patterns discovered by the most frequent seeds

As it can be observed in Figure 23, patterns with different frequencies are discovered by the set of most frequent seed words. Dark colored lines in the tail of the diagram show that low frequency patterns are also being discovered. Low frequency patterns (≤ 3) comprise 45% of the entire discovered patterns (compared to the 63% of the entire set of patterns). This ratio for the set of less-frequent seeds is 35% while the total number of discovered patterns is 12,660.

These results show that the high-frequency seeds, not only co-occur with a larger number of patterns, but also better account for the long tail in the frequency distribution of the patterns compared to lower-frequency seeds.

Chapter 6

Conclusions

In this work, we proposed a corpus-based bootstrapping technique to automatically construct domain-specific semantic lexicons given a small amount of domain knowledge. Our method leverages the collective information from contexts of words, where context is defined as windows of tokens around the instances. Our method requires no training and the only operation needed to be done on the text corpus is identifying phrase boundaries. In addition to a semantic lexicon, our method also learns a dictionary of domain-specific lexical patterns which can be used to extract more category members when applied to other text corpora in the domain or to Web documents. In this chapter, we begin with summarizing our contributions and findings through answering the research questions we raised in section 3.1, and then highlight possible areas for improvement and directions for future work.

Q1: Can lexical patterns effectively capture information about semantic classes of words or do we need richer linguistic information, such as syntactic roles of words obtained from a deep parser?

The results in section 5.5 show that lexical patterns demonstrate superior performance compared to patterns generated from gramamatical dependency relations, in determing the semantic class of nouns/NPs, while they perform somewhat worse in detecting the subjective modifiers (section 5.7). we also observed in section 5.2 that adding syntactic constraints to the lexical patterns does not improve their performance while they limit flexibility of the patterns to extract only nouns/NPs. In fact, a large number of the most effective lexical patterns conform to grammatical relations such as <*subj*> *copula adjective* (e.g., "__ are delicious") or *active-verb* <*dobj*> (e.g., "loved the __"). This means lexical patterns implicitly exploit information about syntactic roles of words in addition to lexical clues syntactic patterns fail to capture. However, the set of rules shown in Table 7 can be augmented with more rules or be tailored for specific tasks.

Given the above observations as well as the cost and inaccuracy of parsing a text corpus, we recommend lexical patterns as a better alternative to syntactic and lexico-syntactic patterns as they are accurate and versatile and allow scalability to larger corpora. However, due to the large number of lexical patterns generated in this way, the performance of a system that employs them largely depends on its ability to learn the most effective patterns (those that capture information about the semantic class of words).

Q2: How do iterative bootstrapping models perform compared to one-pass context vector models?

Both bootstrapping and context vector models are effective methods based on the distributional similarity hypothesis: they hypothesize the semantic class of words based on collective information from their contexts. The former uses the commonality of contexts of known category members and candidate extractions to pinpoint those that belong to the semantic category, while the latter uses the commonality of contexts of seed words and candidate extractions to calculate a semantic similarity score for each candidate.

Bootstrapping models benefit from a broader scope for extraction of category members, since all previously discovered lexicon entries take part in selecting the future entries; as opposed to context vector models where membership in the category is determined by similarity to a fixed set of a few seed words. On the other hand, this makes bootstrapping models more vulnerable to infections (i.e., out-of-category entries which make possible the inclusion of their semantic neighbors in the lexicon). If proper techniques are employed to prevent the drift in the output lexicon, as the results in section 5.6 suggest, bootstrapping models have advantages over context vector models in terms of learning precision.

Q3: What should be chosen as the extraction unit?

Depending on the domain and the task at hand, different constituents[15] might be suitable for inclusion in the semantic lexicon: words (e.g., “pizza” or “delicious”), phrases (e.g., “Italian sausage”), or more complex groups of words (e.g., “fish and chips”). Publicly available sentence analyzers can easily be used to detect word classes and phrase boundaries with reasonable accuracy, while, to detect more complex constituents, more sophisticated and probably less accurate methods should be devised.

[15] A constituent is a word or a group of words that functions as a single unit in the syntax of a sentence.

The decision to pick the best fitting constituent(s) for inclusion in the lexicon remains a tradeoff between the recall of the semantic category and simplicity/accuracy of obtaining them. This tradeoff becomes more favorable to the accuracy and simplicity factors, once one notes that the best fitting constituent(s), even if detected with one hundred percent accuracy, still may not be suitable for inclusion in the semantic lexicon. For example, in the task of IE from opinionated data, NPs are premodified with subjective modifiers which are not part of the valid multi-word terms. Maybe that is why previous works on semantic lexicon induction unanimously use single nouns or heads of NPs as extraction units.

In section 4.4, we proposed methods to automatically identify valid multi-word terms nested within NPs. The least sophisticated and the most effective method was to use a dictionary of subjective adjectives to remove subjective modifiers from NPs. We later showed that our bootstrapping method is flexible enough to be used to acquire such a dictionary automatically or with minimal human validation if a higher accuracy is required.

Q4: How can category drift be prevented in the output semantic lexicon?

We proposed and evaluated three techniques for preventing category drift in the output lexicon. The first two were to learn multiple semantic categories and to use reciprocal feedback between the categories being learned to better constrain and guide the bootstrapping. Employing these techniques had a significant impact on the learning precision (Table 19 and Figure 17). The third technique which was to reduce the contribution of entries learned in later iterations (since they are more likely to be infections), only slightly improved the precision in final iterations.

Learning multiple lexicons, however, may not be as effective or even applicable in every task. Multi-category semantic lexicon construction is most effective where members of two or more different semantic categories in the domain appear in similar contexts (e.g., “dish names” and “aspects of business” in restaurant reviews). In some tasks, the target category is so distinctive that makes it difficult to define a second semantic category to constrain the output lexicon, for example, learning a single lexicon of subjective modifiers.

In order to most effectively prevent the drift in the output lexicon, we believe it is crucial to evaluate the domain and the task beforehand to identify all co-existing semantic categories; and construct semantic lexicons for those that tend to appear in interchangeable contexts.

Q5: What are the effects of the number and composition of seeds on bootstrapping performance?

Based on the results presented in section 5.8, we can conclude with high confidence that increasing the number of seed words even beyond 5 has a negative effect (although minor) on the bootstrapping performance. We select our seed words among the most frequent category members. Due to their high frequency, seed words have great impact on the direction of bootstrapping. We believe providing more seeds results in discovery of entries which are tightly associated with the set of seeds and maybe not with the rest of category members; and therefore stops the bootstrapping from better exploring the search space in the early iterations.

We also observed that a large number of seed words is not required in order to discover infrequent extraction patterns (having corpus frequency of 3 or less). In fact, around 45% of the patterns discovered by the set of 20 most frequent dish names are low frequency patterns, while they constitute only 63% of all patterns in the corpus. Based on these observations, we recommend using small sets of seed words picked from the most frequent members of the target semantic class in the corpus.

6.1 Directions for Future Work

One of the factors that negatively affected the performance of our method is mismatch between the multi-word dish names in the evaluation set and our extraction units (noun phrases). For example, there are many dish names made up of NPs connected with prepositions or conjunctions. NP-chunkers fail to recognize such structures and group the NPs into separate constituents. For example “fish with green curry in banana leaf” would be chunked as “[fish] with [green curry] in [banana leaf]” or “fish and chips” as “[fish] and [chips]”. Our algorithm can greatly benefit from a method that detects the correct boundaries of a compound dish name. For example suffix trees can be used to detect substrings that are repeated in the reviews of a restaurant and heuristics can be applied to filter invalid candidates.

Even with NPs as extraction patterns, we can do a better job of determining the termhood of candidate entries. Frantzi et al. (Frantzi, Ananiadou and Mima 2000) propose the *NC-Value* metric which improves upon *C-value* by incorporating context information from occurrences of candidate terms. Similar ideas can be used for detecting subjective modifiers without relying on an external dictionary of subjective adjectives. For example, instances of valid dish names can be found throughout the corpus co-occurring with various subjective modifiers, while the same dish name when already premodified by a subjective

modifier (e.g., “delicious pizza”) is less likely to co-occur with any other subjective modifier. We suppose simple intuitions like this, if methodized properly, can help detect subjective modifiers.

Although our criteria and methods for the selection and scoring of extraction patterns and candidate entries aim to promote those which are associated the most strongly with the target semantic category, they are far from perfect. For instance, the *RlogF* metric (Eq. 12) used to rank the extraction patterns is biased towards high frequency patterns, while a better ranking scheme should balance high frequency and low frequency patterns.

The right level of generalization can help increase the patterns’ recall without affecting their precision. For example, different linking verbs and their conjugations can replace “tastes” in “__ tastes great”. The precision of extraction patterns may also improve by adding certain semantic constraints. Devising methods to combine multiple syntactic relations into one extraction pattern in order to increase the amount of information it conveys and consequently increase its extraction power can also be an interesting future work.

(Igo and Riloff 2009) report significant improvements upon Basilisk by re-ranking the final lexicon based on Web co-occurrence statistics between lexicon entries and seed words (they use PMI as the measure of co-occurrence). They propose, but do not implement, incorporating this Web-based re-ranking procedure into the bootstrapping algorithm itself to re-rank the candidate extraction before they are selected for addition to the lexicon. This idea is promising because it could improve the precision of the lexicon in early iterations, when the bootstrapping is most vulnerable to infections. It is also interesting to experiment using statistics drawn from the corpus along with co-occurrence measures such as likelihood ratio which are more suitable for corpus-based tasks.

And finally, another idea that can effectively augment the output lexicon with more valid category members, is to locate lists in the corpus or in Web pages, where category members co-occur. For example, we can look for co-occurrences of lexicon entries in HTML tables where they have identical formattings, and augment the lexicon with other terms in the table that share the same formatting. The retionale is that if a high percentage of elements in a list are among known category members, there is a high chance that the rest of the elements also belong to the same semantic category. In order to not risk the precision, we can assign a confidence score to every list, based on what fraction of its elements are known category members, and select new terms only from high confidence lists.

Glossary

NLP – *Natural Language Processing,* a sub-field of Artificial Intelligence which studies problems of computationally understanding human languages.

IR – *Information Retrieval,* the area of study concerned with searching for documents, information within documents, and metadata about documents.

PMI – *Pointwise Mutual Information,* a measure of association used in information theory and statistics.

WSD – *Word-sense Disambiguation,* the process of identifying which sense (meaning) of a word is used in a sentence, when the word has multiple meanings (polysemy).

POS – *Part of Speech,* the linguistic category of words, which is generally defined by the syntactic or morphological behavior of the word in question.

BM25 – a ranking function used to rank documents according to their relevance to a given search query.

QACW – *Query Adjusted Combined Weight,* a weighting function used in probabilistic models of Information Retrieval.

NP(s) – *Noun Phrase(s),* a phrase based on a noun, pronoun, or other noun-like words (nominals), optionally accompanied by modifiers such as adjectives.

GN(P) – *General Noun (Phrase),* any noun/non phrase not representing a unique entity.

PN(P) – *Proper Noun (Phrase),* a noun/non phrase representing a unique entity.

VP – *Verbal Phrase,* a syntactic unit composed of at least one verb and the dependents of that verb.

NER – *Named-entity Recognition,* a subtask of information extraction that seeks to locate and classify atomic elements in text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

RE – *Relation Extraction,* the task of extracting semantic relations between entities in text.

TREC – *Text REtrieval Conference,* focuses on different IR research areas. Its purpose is to support and further research in the IR discipline.

REF – *Related Entity Finding,* a task in the Entity Track of TREC, whose goal is to find entities that are of a target type and stand in the required relation to an input entity.

QA – *Question Answering,* the task of automatically answering a question posed in natural language.

CRF – *Conditional Random Fields,* a class of statistical modeling method often applied in pattern recognition and machine learning for structured prediction.

IE – *Information Extraction,* type of information retrieval whose goal is to automatically extract structured information from unstructured or semi-structured machine-readable documents.

NE – *Named-entity,* a named entity is an element in text that clearly identifies one item from a set of other items that have similar attributes.

IAA – *Inter-Annotator Agreement,* the degree of agreement in the annotations given by judges.

MWU – *Multi-word Unit,* a special type of collocate in which the component words comprise a meaningful phrase.

AveP – *Average Precision,* Average of the precision values at the points at which each relevant document is retrieved.

MAP – *Mean Average Precision,* is a single-valued measure of quality across recall levels, defined as the Mean of Average Precision values in various trials.

References

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

Balog, K, P Serdyukov, and A de Vries. "Overview of the TREC 2010 Entity Track." *TREC 2010.* 2010.

Banko, M, and O Etzioni. "The Tradeoffs Between Open and Traditional Relation Extraction." *the Association of Computational Linguistics.* 2008.

Brin, S. "Extracting Patterns and Relations from the World Wide Web." *WebDB Workshop at 6th International Conference on Extending Database Technology.* Valencia, Spain, 1998. 172-183.

Caraballo, S. "Automatic construction of a hypernym-labeled noun hierarchy from text." *the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics.* College Park, Ma, 1999. 120-126.

Church, K, W Gale, and P Hanks. "Using statistics in lexical analysis." In *In Lexical Acquisition: Exploiting On-Line Resources to Build a Lexicon*, 115-164. New Jersey: Lawrence Erlbaum, 1991.

Cimiano, P, and J Volker. "Towards large-scale, open-domain and ontology-based named entity classification." *RANLP'05.* Borovets, Bulgaria, 2005. 166-172.

Davidov, D, and A Rappoport. "Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words." *the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics.* Sydney, Australia, 2006. 297-304.

De Marneffe, M-C, B MacCartney, and C D Manning. "Generating Typed Dependency Parses from Phrase Structure Parses." *the Fifth International Conference on Language Resources and Evaluation (LREC'06).* 2006. 449-454.

Dorow, B, D Widdows, K Ling, J Eckmann, D Sergi, and E Moses. "Using Curvature and Markov Clustering in Graphs for Lexical Acquisition and Word Sense Discrimination." *MEANING '05.* 2005.

Dunning, T. "Accurate methods for the statistics of surprise and coincidence." *Computational Linguistics*, 1993: 61-74.

Etzioni, O, et al. "Unsupervised named-entity extraction from the Web: An experimental study." *Artificial Intelligence*, 2005: 91-134.

Fleischman, M, and E Hovy. "Fine Grained Classification of Named Entities." *Proceedings of the 19th international conference on Computational linguistics (COLING '02).* Taipei, Taiwan, 2002.

Frakes, W, and R Baeza-Yates. *Information Retrieval, Data Structures and Algorithms.* Prentice Hall, 1992.

Frantzi, K, and S Ananiadou. "Extracting nested collocations." *the 16th conference on Computational linguistics.* Copenhagen, Denmark, 1996.

Frantzi, K, S Ananiadou, and H Mima. "Automatic recognition of multi-word terms: the C-value/NC-value method." *International Journal on Digital Libraries*, 2000: 115-130.

Grefenstette, G. *Explorations in automatic thesaurus discovery.* Kluwer, 1994.

Harris, Z. "Distributional structure." *Word* 10, no. 23 (1954): 146-162.

Hearst, M. "Automatic acquisition of hyponyms from large text corpora." *the 14th conference on Computational linguistics (COLING '92).* 1992.

Hogue, A. *The Essentials of English: A Writer's Handbook.* Longman Pub Group, 2003.

Hu, M, and B Liu. "Mining and summarizing customer reviews." *the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* Seattle, WA, USA, 2004.

Igo, S.P., and E Riloff. "Corpus-based semantic lexicon induction with Web-based corroboration." *the Workshop on Unsupervised and Minimally Supervised Learning of Lexical Semantics.* Boulder, Colorado, 2009. 18-26.

Kozareva, Z, E Riloff, and E Hovy. "Semantic Class Learning from the Web with Hyponym Pattern Linkage Graph." *ACL-08.* 2008. 1048-1056.

Lafferty, J, A McCallum, and F Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." *the Eighteenth International Conference on Machine Learning.* 2001.

Lenat, D.B., M Prakash, and M Shepherd. "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks." *AI Magazine*, 1985: 65-85.

Lin, D. "Automatic retrieval and clustering of similar words." *the 17th international conference on Computational linguistics (COLING '98).* 98. 768-774.

Lin, D. "Principle-Based Parsing Without OverGeneration." *ACL-93.* Columbus, OH, 1993. 112-120.

Lin, D. "Using syntactic dependency as local context to resolve word sense ambiguity." *the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics.* Madrid, Spain, 97. 64-71.

Lin, D, and P Pantel. "Concept Discovery from Text." *the 19th international conference on Computational linguistics.* Taipei, Taiwan, 2002. 1-7.

Lin, W, R Yangarber, and R Grishman. "Bootstrapped learning of semantic classes from positive and negative examples." *the ICML Workshop on The Continuum from Labeled to Unlabeled Data.* Washington, D.C, 2003.

Liu, J, X Li, A Acero, and Y-Y Wang. "Lexicon Modeling For Query Understanding." *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* Prague, 2011. 5604-5607 .

Manning, C, and H Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

McCallum, A, and W Li. "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons." *the seventh conference on Natural language learning at HLT-NAACL.* Edmonton, Canada, 2003. 188-191.

Miller, G.A., R Beckwith, C Fellbaum, D Gross, and K.J. Miller. "Introduction to WordNet: An On-line Lexical Database." *International Journal of Lexicography*, 1990: 235-244.

Pantel, P, and D Ravichandran. "Automatically labeling semantic classes." *HLT/NAACL-04.* Boston, MA, 2004. 321-328.

Pantel, P, E Crestan, A Borkovsky, A Popescu, and V Vyas. "Web-scale distributional similarity and entity set expansion." *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP '09).* Singapore, 2009.

Paşca, M. "Acquisition of categorized named entities for web search." *the thirteenth ACM international conference on Information and knowledge management.* 2004. 137-145.

Paşca, M, and B, J Durme. "Weakly-supervised Acquisition of Open-Domain Classes and Class Attributes from Web Documents and Query Logs." *ACL-08.* 2008.

Phillips, W, and E Riloff. "Exploiting Strong Syntactic Heuristics and Co-Training to Learn Semantic." *the Conference on Empirical Methods in Natural Language Processing (EMNLP).* Philadelphia, 2002. 125-132.

Ravichandran, D, and E Hovy. "Learning Surface Text Patterns for a Question Answering System." *the 40th Annual Meeting of the Association for Computational Linguistics (ACL).* Philadelphia, 2002. 41-47.

Riloff, E. "An empirical study of automated dictionary construction for information extraction in three domains." *Artificial Intelligence* 85 (1996).

Riloff, E. "Automatically generating extraction patterns from Untagged Text." *the Thirteenth National Conference on Artificial Intelligence.* 1996. 1044–1049.

Riloff, E, and J Shepherd. "A corpus-based approach for building semantic lexicons." *Second Conference on Empirical Methods in Natural Language Processing.* Brown University Providence, Rhode Island, USA, 1997. 117-124.

Riloff, E, and R Jones. "Learning dictionaries for information extraction by multi-level bootstrapping." *the 16th National Conference on Artificial Intelligence.* 1999.

Riloff, E, J Wiebe, and T Wilson. "Learning subjective nouns using extraction pattern bootstrapping." *the seventh conference on Natural language learning at HLT-NAACL 2003.* Edmonton, Canada, 2003. 25-32.

Roark, B, and E Charniak. "Noun-phrase co-occurrence statistics for semiautomatic semantic lexicon construction." *the 17th international conference on Computational linguistics.* Stroudsburg, PA, USA, 1998. 1110-1116.

Robertson, S, S Walker, S Jones, M Hancock-Beaulieu, and M Gatford. "Okapi at TREC-3. InHarman D. (Ed.) ." *the Third Text Retrieval Conference (TREC).* Gaithersburg, MD, United States, 1995. 109-126.

Spärck Jones, K, S Walker, and S E Robertson. "A probabilistic model of information retrieval: Development and comparative experiments." *Information Processing and Management* 6, no. 36 (2000): 779–808 (Part 1); 809–840 (Part 2).

Tan, P, V Kumar, and J Srivastava. "Selectingthe right interestingness measure for association patterns." *the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* 2002. 32-41.

Terra, E, and C Clarke. "Frequency estimates for statistical word similarity measures." *the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, NAACL '03.* Edmonton, Canada, 2003. 165-172.

Thelen, M, and E Riloff. "A bootstrapping method for learning semantic lexicons using extraction pattern contexts." *the ACL-02 conference on Empirical methods in natural language processing.* 2002.

Tsai, R, and C Chou. "Extracting dish names from chinese blog reviews using suffix arrays and a multi-modal CRF model." *the first International Workshop on Entity-Oriented Search (EOS).* 2011. 7-13.

Turney, P. "Mining the Web for Synonyms: PMI-IR Versus LSA on TOEFL." *the Twelfth European Conference on Machine Learning (ECML-2001).* Freiburg, Germany, 2001.

Vechtomova, O, and S.E. Robertson. "A domain-independent approach to finding related entities." *Information Processing and Management*, 2011.

Vechtomova, O, M.H. Ahmadi, and K Suleman. "Building Fine-Grained Semantic Classes: a Semi-Supervised Approach." *Technical Report, University of Waterloo.* 2012.

Wang, Y, R Hoffmann, X Li, and J Szymanski. "Semi-supervised learning of semantic classes for query understanding: from the web and for the web." *the 18th ACM conference on Information and knowledge management.* Hong Kong, China , 2009.

Welty, C, J Fan, D Gondek, and A Schlaikjer. "Large scale relation detection." *the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading.* 2010.

Whitelaw, C, A Kehlenbeck, N Petrovic, and L Ungar. "Web-scale named Entity Recognition." *17th ACM conference on Information and knowledge mining (CIKM 08).* Napa Valley, California, 2008. 26-30.

Widdows, D, and B Dorow. "A graph model for unsupervised lexical acquisition." *Proceedings of the 19th international conference on Computational linguistics (COLING 02).* 2002. 1-7.

Wiebe, J, T Wilson, and C Cardie. "Language Resources and Evalution (formerly Computers and the Humanities)." *Annotating expressions of opinions and emotions in language.*, 2005.

Yarowsky, D. "Word sense disambiguation usingstatistical models of Roget's categories trainedon large corpora." *the Fourteenth International Conference on Computational Linguistics(COLING-92).* 1992. 454-460.