# Information Extraction from Unstructured Web Text

Ana-Maria Popescu

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2007

Program Authorized to Offer Degree: Department of Computer Science and Engineering

University of Washington

## Abstract

Information Extraction from Unstructured Web Text

Ana-Maria Popescu

Chair of the Supervisory Committee:
Professor Oren Etzioni
Department of Computer Science and Engineering

In the past few years the Word Wide Web has emerged as an important source of data, much of it in the form of unstructured text. This thesis describes an extensible model for information extraction that takes advantage of the unique characteristics of Web text and leverages existent search engine technology in order to ensure the quality of the extracted information. The key features of our approach are the use of lexico-syntactic patterns, Web-scale statistics and unsupervised or semi-supervised learning methods. Our information extraction model has been instantiated and extended in order to solve a set of diverse information extraction tasks: subclass and related class extraction, relation property learning, the acquisition of salient product features and corresponding user opinions from customer reviews and finally, the mining of commonsense information from the Web for the benefit of integrated AI systems.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

## ACKNOWLEDGMENTS

This thesis could not have been completed without the patient guidance of my advisor, prof. Oren Etzioni, the help and comments of the KnowItAll research group and the support of my family. I would also like to thank the Computer Science Department and especially prof. Alon Halevy and prof. Dan Weld whose feedback has been most appreciated.

# Chapter 1
# INTRODUCTION

In recent years the World Wide Web has emerged as an important and dynamic source of information which has attracted the attention of researchers from many areas of artificial intelligence (AI) and natural language processing (NLP): information extraction [57, 35], question answering [58, 69], intelligent agents [55, 8], ontology building for the Semantic Web [63] and many others.

The Web contains a wealth of information, much of it available in the form of *unstructured text*. Information Extraction (IE) is an active area of research which is concerned with extracting information about types of events, entities or relationships from textual data. While information extraction (as well as related fields such as information retrieval and information synthesis [10]) has successfully used news corpora, domain-specific text corpora (*e.g.*, medical literature), manuals or dictionaries, recent IE efforts have started exploring the Web as a source of textual information. Initial Web-based information extraction focused on exploiting *structured* and *semi-structured text* (e.g., [57, 5, 105]). Recently, researchers have also turned their attention towards *unstructured Web text* such as product and hotel reviews, newsgroup postings, how-to sites and many other types of textual data.

## *1.1 Characteristics of Unstructured Web Text*

As detailed in [58, 35, 90], *unstructured* Web text offers a number of advantages over previously used text collections:

- **Redundancy** Facts seldom mentioned in a news corpus may be mentioned often on the Web.

- **Multiple paraphrases** Facts that are often mentioned on the Web are more likely to appear in a variety of formulations (or *paraphrases*).

- **Easy-to-understand language** Since facts are likely to be repeatedly paraphrased, some paraphrases will be easier to understand than others. Previously used text corpora tended to consist of newspaper articles whose language is more formal and contains complicated linguistic and syntactic constructions; retrieving information from such articles requires relatively sophisticated methods, whereas Web text is more likely to contain simpler constructions that can be handled using a less complicated linguistic processing machinery.

- **Broad coverage** The Web contains many types of useful information: product, restaurant and hotel reviews, how-to sites with large quantities of commonsense information, newspaper articles and so on. The variety of the information available on the Web supports many new types of applications (*e.g.*, recommender and trend detection systems, integrated AI applications such as activity recognition and state estimation systems, and more).

- **Search Engines** Search engines offer an easy way to navigate Web text, retrieve information and collect counts for strings of interest.

We mentioned the advantages of using unstructured Web text as a source of information, but this type of Web text also presents a number of challenges (see [58, 35]):

- **Unreliable information** A well-known shortcoming of the Web is the presence of unreliable information: unlike manuals or corpora of newspaper articles, the Web contains information from potentially unreliable or biased sources. While a frequently mentioned piece of information is typically more likely to be correct than an item mentioned once or twice, frequency is not a guarantee of truth.

- **Ungrammatical language** Unlike newspaper text, Web text can be colloquial, unedited and therefore ungrammatical. Deep parsing of such text may be problematic while more lightweight linguistic processing methods are likely to work better.

## 1.2 Thesis Overview

In this thesis, we investigate the following hypothesis: *unstructured Web text can be used to extract useful, diverse, high quality information.*

### 1.2.1 Thesis Contribution

In order to investigate this hypothesis, we use a simple and extensible model for information extraction that takes advantage of the unique characteristics of Web text and leverages existent search engine technology in order to ensure high precision. The key features of our approach are the use of simple extraction patterns, Web co-occurence statistics and unsupervised or semi-supervised learning methods. Furthermore, we show that the basic information extraction model can be instantiated and extended to solve a number of interesting and important tasks.

This document describes the aforementioned information extraction model together with a set of case studies. The rest of this document is organized as follows:

- The remainder of Chapter 1 describes the core information extraction model and contains a summary of the corresponding case studies in which the model was used; the chapter ends with an overview of related work.

- Chapter 2 contains a description of the KnowItAll information extraction system, on which our information extraction architecture is based.

- Chapters 3 through 6 describe in detail the case studies we addressed and show that the ideas behind our information extraction architecture are applicable to a variety of information extraction scenarios.

## 1.3 WIE: *A Web-based Information Extraction Architecture*

The information extraction architecture described in this section is an augmented version of the architecture of a state-of-art information extraction system, KnowItAll [35, 34].

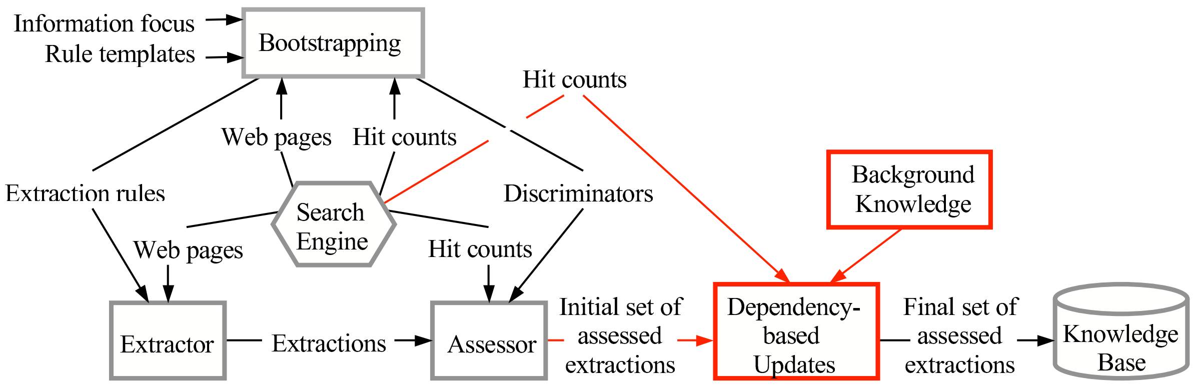


Figure 1.1: **The** WIE **Architecture for Web-based Information Extraction.** The Bootstrapping module creates extraction rules for predicates of interest; the Extractor uses them to generate extractions that are validated by the Assessor using search engine hit counts. The resulting probabilities can be refined using dependencies among extractions; the dependencies are obtained from Web text or from background knowledge. The final set of assessed extractions is then added to a knowledge base.

KnowItAll, which is described in detail in Chapter 2, is a Web-based information extraction system that can extract large numbers of high-quality instances of classes (*e.g.*, instances of the class `City`) or relations (*e.g.*, instances of `capitalOf(City, Country)`).

Figure 1.1 gives an overview of WIE, an extended version of KnowItAll's architecture; the *Dependency-based Updates* and *Background Knowledge* components are additions to the basic KnowItAll architecture. In the following, we give an overview of WIE and then describe the added components in more detail.

WIE takes as input a set of predicates which specify its focus: *e.g.*, {`Scanner`,`Printer`, `Phone`, `Computer`} is a set of concepts of interest to a buyer of consumer electronics.

For each user query (*e.g.*, "find instances of predicate P"), the *Bootstrapping* step uses a set of manually supplied *domain-independent* extraction patterns (*e.g.*, Figure 1.2) to create a set of extraction rules and "discriminator" phrases (described below) corresponding to the predicate of interest. The Bootstrapping is fully automatic, in contrast to other bootstrapping methods that require a set of manually created training seeds.

| | |
|---|---|
| Predicate: | Class1 |
| Pattern: | NP1 "such as" NPList2 |
| Constraints: | head(NP1)= plural(label(Class1)) & properNoun(head(each(NPList2))) |
| Bindings: | Class1(head(each(NPList2))) |

Figure 1.2: **Extraction Rule Template Example.** A generic extraction pattern can be instantiated automatically with the pluralized class label to create a domain-specific extraction rule. For example, if `Class1` is set to "City" then the rule looks for the words "cities such as" and extracts the heads of the proper nouns following that phrase as potential cities.

"the X scanner " " X is a scanner " " scanners such as X"

Figure 1.3: **Discriminator Examples.** Examples of discriminators used to assess candidate instances of the Scanner class.

The Bootstrapping module instantiates the provided rule templates with the name of each predicate of interest and thus generates a set of predicate-specific *extraction rules.* It then uses these rules to bootstrap a small set of answers to the query of interest.

The bootstrapped answers are used to find a set of high precision rules we call *discriminators* (see Figure 1.3 for some examples). The discriminators are used by the Assessor to assign a probability to each extraction and are trained using the bootstrapped set of seed extractions. WIE assesses the plausibility of each candidate fact by computing *pointwise mutual information (PMI) statistics* using the Google search engine.

In addition to the basic assessment step inherited from the KnowItAll architecture, WIE has the option of using a second assessment step that refines the probabilities assigned to candidate facts based on various types of *dependencies* among such facts. These dependencies are found or verified by using either Web statistics or information from available knowledge sources (*e.g.*, WordNet). Introducing a second assessment step results in improvements in the system's perfomance; it can also decrease the number of search engine queries necessary for fact assessment.

### *1.3.1 Dependency-based Updates of Initial Extraction Probabilities*

The set of extractions with associated initial probabilities is taken as input by the *dependency-based update module* that uses various dependencies among extractions to compute the final version of their associated probabilities. Sometimes (*e.g.*, in the case of the review mining system described in Chapter 5), using the Assessor in order to compute a set of initial probabilities for each extraction can be expensive - in such cases, the Assessor can be used for a subset of potential extractions and the dependency-based update module can compute the probabilities associated with the remainder of the potential extractions.

The information extraction tasks described in the following chapters make use of various types of dependencies that can be obtained from Web documents or from additional background knowledge resources (*e.g.*, WordNet [73]). Examples of dependencies among extractions include similarity relationships based on synonymy, antonymy, morphological and context-dependent co-occurrence information.

As we will see in the next chapters, the specifics of the probability refinement mechanisms differ based on the details of the information extraction task at hand, but they are all *unsupervised* or *semi-supervised* methods. For example, our review mining chapter investigates the use of the well-known *relaxation labeling* framework (common in machine vision applications) for the purpose of finding the context-dependent semantic orientation of opinion words. The thesis chapters report on results concerning the use of dependency-based updates for this labeling task, as well as for the tasks of *subclass* and *related class extraction*; additionally, we are in the process of investigating the impact of dependency-based updates on improving the performance of our results on the tasks of *relation property learning* as well as on the tasks of *assessing* and *acquiring commonsense knowledge*.

In the following, we give a brief summary of the information extraction scenarios we investigated and describe the current results for each research effort.

### *1.3.2 High Precision Information Extraction from Web Text: Case Studies*

We instantiated and extended the information extraction architecture described above in order to address a varied set of information extraction tasks briefly described below:

- **Class Extraction** Chapter 3 describes the use of our IE model in the acquisition of high precision *subclasses* of a given class, as well as in the acquisition of other *related classes* [92]. Our results show that subclass extraction can boost the yield of class instance extraction from the Web by a factor of up to 10 [35]. We also describe our promising preliminary experiments in employing a version of the above extraction model in order to expand the WordNet IS-A hierarchy using Web data [89].

  This chapter shows that our IE model can be readily instantiated and used for discovering high precision subclass and related class information from the Web, which demonstrates the model's usefulness and generality.

- **Learning Relation Properties** Chapter 4 describes our preliminary work on learning relation properties (*e.g.*, *transitivity*) and ontologically relevant dependencies (*e.g.*, *entailment*) among relations (*under preparation*). Such information is likely to be useful for a variety of tasks, such as fact extraction and deduplication. The main ideas on which our IE model relies (Web-scale statistics for assessment and the use of lexico-syntactic patterns) are employed with good results in the building of a property learner, further demonstrating their value. On-going work on this problem includes the use of the dependency-based update module as part of a collective assessment scheme in order to improve the system's performance; we are in the process of defining and testing different types of dependencies among relations as well as among meta-properties of interest.

- **Mining Product Features and Corresponding Opinions** The Web contains a wealth of opinions about products, politicians and more, which are expressed in newsgroup posts, review sites, and elsewhere. Chapter 5 describes our work on high precision extraction of *product features* and corresponding *user opinions* from product reviews. We automatically determine the positive or negative *context-dependent semantic orientation* of each opinion as well the *relative strength* of opinions that refer to the same property (*e.g.*, "spotless" is a stronger endorsement than "clean" when discussing Cleanliness). This detailed opinion information is used to generate opinion summaries of customer reviews [90, 91].

Our review mining system consists of a number of components which rely on our IE model. Web-scale statistics for assessment lead to improved feature extraction, lexico-syntactic patterns are used for feature acquisition, finding the semantic orientation of opinion words and phrases as well as determining the relative strength of opinions; finally, dependency-based probability updates allow us to find the semantic orientation of opinion words and phrases with high precision.

- **Mining Commonsense Relationships** The Web also contains a wealth of commonsense knowledge, which can finally be exploited by activity recognition and state estimation systems due to improvements in sensor technology and advances in probabilistic inference. Chapter 6 shows that the Web can be used to validate existent commonsense information provided by volunteers and that the Web-validated data leads to improvements in the performance of a state-of-the-art state estimation system [86]. We also show how our extract-and-assess model can be adapted to the task of acquiring relations among commonsense state and action events for the purpose of partially replacing an existent commonsense knowledge base (*in preparation*). Ongoing work in this area includes adapting and testing our dependency-based update module in order to improve the system's performance on the tasks of assessing and acquiring commonsense knowledge.

In conclusion, we describe an extensible, Web-based information extraction model and show how it can be adapted and used with good results in a variety of information extraction scenarios.

## *1.4 Related Work*

This section contains an overview of the key areas of related work - each subsequent chapter contains a discussion of additional research papers directly related to the information extraction task at hand.

### *1.4.1 The Use of co-occurrence Statistics and the Use of the Web as a Corpus*

co-occurrence statistics computed from scratch over document collections have been extensively used for natural language processing tasks such as synonym recognition, lexicon construction and other lexical analysis and text mining tasks [19, 20, 98]. While co-occurrence statistics have been traditionally used in the context of unstructured text corpora, recent research has started importing such techniques into the world of structured data by computing statistics over a corpus of structures [45].

Turney [114] was the first to realize that commercial search engines offer a cheap way to compute Web-scale co-occurrence statistics; his PMI-IR algorithm uses search engine hit counts to compute the pointwise mutual information (PMI) between two words. This correlation measure is used to find a phrase's semantic orientation by comparing it with strongly positive or strongly negative words (*e.g.*, "great", "awful"). Other researchers (*e.g.*, [118, 64, 18]) have made use of Web statistics in order to validate answers to questions, instances of a given class or semantic information such as verb entailment.

The KnowItAll project was the first large-scale information extraction system to leverage Turney's original insight and conclusively show that Web statistics can ensure the high quality of extracted information. Inspired by our results, other researchers have recently used Web co-occurrence statistics for IE: [68] has looked at populating the Cyc knowledge base with instances of binary relations extracted from the Web while [75, 77, 76] have looked at using Web data to find names for noun-noun relations and aid with compound noun bracketing - the authors also show that search engine hit counts are helpful even though they vary across search engines; finally, [125] added to the list of uses for Web-scale statistics by showing that they can be employed to detect parser errors.

As we will see in the following chapters, this thesis shows that co-occurrence statistics at Web scale can be used to validate various types of extracted information: subclass extractions, related concepts, relation meta-properties, product features, instances of commonsense relationships and more. In certain cases (*e.g.*, extraction of subclass terms, related concepts, product features), this thesis is the first to report experimental results concerning the use of Web-scale co-occurrence statistics for the task at hand. In other cases (*e.g.*, learning the semantic orientation of opinion words and phrases), this thesis uses co-occurrence statistics as part of new ways of solving the given task.

### *1.4.2 Information Extraction from Unstructured Text*

The work described in this thesis leverages and extends some ideas explored by previous systems for information extraction from unstructured text [88, 95, 96, 52, 79, 2, 13, 24]: the use of simple lexico-syntactic patterns [48] in the generation of extraction rules and the use of a bootstrap learning approach (alternating between learning rules or rule accuracy from sets of instances and finding instances using sets of rules).

In this thesis, we confirm the wide applicability of such previously introduced techniques by using an architecture that combines, updates and applies them to a number of interesting and varied applications. The past year (2006) has seen a renewed interest in such simple, lightweight systems that use generic patterns and a bootstrap approach in order to learn semantic relations from text (*e.g.*, [84, 85]). Also in recent months, [7] and [100] have recommended moving towards *open information extraction* systems, which are not limited by a set of initial predicates but instead extract a large number of facts and afterwards identify the underlying semantic relations.

### *1.4.3 Dependency-based Probability Updates*

A key component of our architecture is the second assessment step that uses dependencies among the extractions in order to refine the probabilities assigned by the first assessment step or "propagate" label information to additional potential extractions. While our work was independently completed, parallel (and subsequent) research efforts used techniques with a similar flavor to address other interesting text mining tasks: [14] introduced a collective information extraction system developed in a Relational Markov Network framework that focused on named entity classification and de-duplication tasks in the context of bio-medical literature, [80] used *graph mincuts* to label review data as positive or negative while [110] used a *spin model* to automatically label a large number of words with context-independent semantic orientation labels; [17] employed the recently introduced *label prop-*

*agation* mechanism to label binary relation instances with the appropriate relation name; [104] developed a probabilistic model for combining information from multiple sources in order to learn semantic IS-A hierarchies from text data.

As previously mentioned, the above papers focus on text mining tasks differing from ours or use different techniques than the ones used in this thesis. Our work adds to the body of literature that shows that dependency-based probability updates can be useful for a variety of tasks and can be accomplished using a variety of techniques (in our case, unsupervised and semi-supervised techniques).

A related class of systems that make use of various types of dependencies are systems for *multifield information extraction* (*e.g.*, [65, 38, 121, 72]), which find instances of n-ary predicates from text corpora. In the last chapter of this thesis we outline our work in this vein for extracting instances of commonsense relations from Web text.

Using complex dependencies among entities or relations has been explored in depth outside of the text mining literature as well; [109, 102, 103, 71] address tasks such as deduplication, also known as "record linkage", which occurs in a large number of applications, especially database applications.

### *1.4.4 Constructing Ontologies from Text*

While much of the early information extraction research was conducted in the context of the MUC challenges, similar research problems were confronted in parallel by researchers interested in building or refining ontologies using textual information. Many papers (*e.g.*, [37, 41, 62, 60]) have addressed the acquisition of classes, relations and relation types from text in order to build a domain ontology from scratch or in order to augment or refine an already existent ontology.

Building, refining and matching [29] ontologies has become even more important in the context of the Semantic Web, which depends on their proliferation and management [63]. Ontologies can be used for manual or automatic creation of semantically interlinked metadata (as exemplified by the semantic annotation efforts described in [32, 28], among others).

In the remaining chapters of this thesis we show that Web text can be used to enrich ontologies in a number of ways: by extracting subclasses, features and related concepts for given target concepts, by automatically verifying relation meta-properties and by populating common sense knowledge bases with instances of commonsense relations.

# Chapter 2
# AN OVERVIEW OF THE KNOWITALL INFORMATION EXTRACTION SYSTEM

## 2.1 Introduction

KnowItAll [35, 34] is a Web-based information extraction system with a novel *extract-and-assess* architecture that uses generic lexico-syntactic patterns and Web-scale statistics to extract *class* and *relation instances* from unstructured text on the Web.

In its first major run KnowItAll extracted over 50,000 class instances corresponding to three predicates of interest: `City`, `Country` and `Film`. In subsequent work [35, 34] our group has explored three ways of improving the basic system's recall while maintaining its high precision: *Pattern Learning* learns domain-specific extraction rules, which enable additional extractions; *Subclass Extraction* automatically identifies subclasses in order to boost recall (*e.g.*, "chemist" and "biologist" are identified as subclasses of "scientist"); *List Extraction* locates lists of class instances, learns a "wrapper" for each list, and extracts elements of each list.

In concert, our methods gave KnowItAll a 4-fold to 8-fold increase in recall at precision of 0.90, and discovered over 10,000 cities missing from the Tipster Gazetteer.

The remainder of this background chapter is organized as follows: we start by describing the architecture of the basic KnowItAll system, which is the basis of the WIE architecture at the core of this thesis; we then give an overview of the relevant related work.

## 2.2 KnowItAll *: An Architecture for Information Extraction from the Web*

KnowItAll is a Web-based information extraction system [35, 34] that can extract large numbers of high-quality instances of classes (e.g., instances of the class `City`) or relations (e.g., instances of `capitalOf(City, Country)`).

In the following, we give an overview of the KnowItAll architecture and then describe its components in more detail - the following description is a modified version of the basic system description in our 2005 journal paper [34].

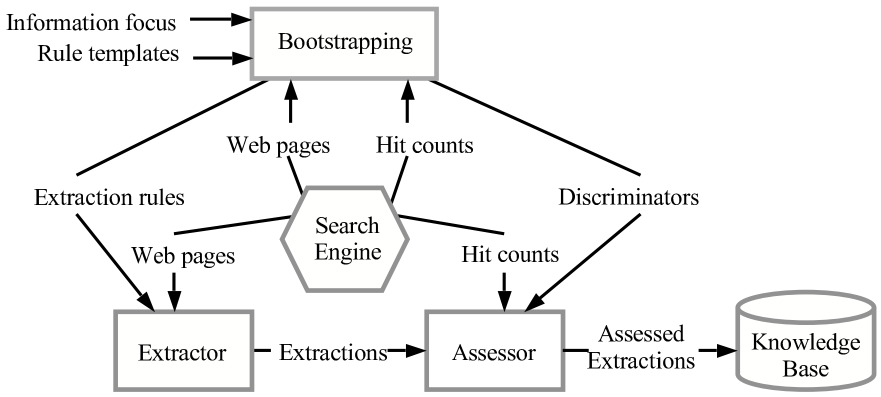


Figure 2.1: **A Basic Architecture for Web-based Information Extraction:** KnowItAll The Bootstrapping module creates extraction rules for predicates of interest; the Extractor uses them to generate extractions that are validated by the Assessor using search engine hit counts. The assessed extractions are then added to a knowledge base.

KnowItAll takes as input a set of predicates which specify its focus; KnowItAll's *Bootstrapping* step uses a set of *domain-independent* extraction patterns (*e.g.*, Figure 2.2) to create a set of extraction rules and "discriminator" phrases (described below) for each predicate in its focus. The Bootstrapping is fully automatic, in contrast to other bootstrapping methods that require a set of manually created training seeds.

The Bootstrapping module instantiates the provided rule templates with the name of each predicate of interest and thus generates a set of predicate-specific *extraction rules.* It then uses these rules to bootstrap a small set of answers to the query of interest.

The bootstrapped answers are used to find a set of high precision rules we call *discriminators* (see Section 2.2.2 for some examples).

The discriminators are used by the Assessor to assign a probability to each extraction and are trained using the bootstrapped set of seed extractions. KnowItAll assesses the plausibility of each candidate fact by using *pointwise mutual information (PMI) statistics* computed using the Google search engine.

In the following, we describe each of the architecture components in more detail.

### *2.2.1 Extractor Description*

| | |
|---|---|
| Predicate: | Class1 |
| Pattern: | NP1 “such as” NPList2 |
| Constraints: | head(NP1)= plural(label(Class1)) & |
| | properNoun(head(each(NPList2))) |
| Bindings: | Class1(head(each(NPList2))) |

Figure 2.2: **Extraction Rule Template Example.** A generic extraction pattern can be instantiated automatically with the pluralized class label to create a domain-specific extraction rule. For example, if `Class1` is set to “City” then the rule looks for the words “cities such as” and extracts the heads of the proper nouns following that phrase as potential cities.

| | |
|---|---|
| Predicate: | City |
| Pattern: | NP1 “such as” NPList2 |
| Constraints: | head(NP1)= “cities” |
| | properNoun(head(each(NPList2))) |
| Bindings: | City(head(each(NPList2))) |
| Keywords: | “cities such as” |

Figure 2.3: **Extraction Rule Example.** An extraction rule generated by substituting the class name City and the plural of the class label “city” into a generic rule template. The rule looks for Web pages containing the phrase “cities such as” and extracts the proper nouns following that phrase as instances of the unary predicate `City`.

The Extractor uses the set of extraction rules provided by the Bootstrap module to find a set of candidate answers to each predicate-specific query.

*Extraction Rules*

The extraction rules are used to formulate search engine queries - the Extractor downloads matching Web pages and applies the extraction rules to each page in order to find the candidate extractions.

KnowItAll automatically creates a set of extraction rules for each predicate, as described in Section 2.2.2. Each rule consists of a predicate, an extraction pattern, constraints, bindings, and keywords. The *predicate* gives the relation name and class name of each predicate argument. In the rule shown in Figure 2.3, the unary predicate is "City". The *extraction pattern* is applied to a sentence and has a sequence of alternating context strings and *slots*, where each slot represents a string from the sentence. The rule may set constraints on a slot, and may bind it to one of the predicate arguments as a phrase to be extracted. In the example rule, the extraction pattern consists of three elements: a slot named NP1, a context string "such as", and a slot named NPList2. There is an implicit constraint on slots with name NP<digit>. They must match simple noun phrases and those with name NPList<digit> match a list of simple noun phrases. Slot names of P<digit> can match arbitrary phrases.

The Extractor uses regular expressions based on part-of-speech tags from the Brill tagger [12] to identify simple noun phrases and NPLists. The head of a noun phrase is generally the last word of the phrase. If the last word is capitalized, the Extractor searches left for the start of the proper noun, based on orthographic clues. Take for example, the sentence "The tour includes major cities such as New York, central Los Angeles, and Dallas". The head of the NP "major cities" is just "cities", whereas the head of "New York" is "New York" and the head of "central Los Angeles" is "Los Angeles". This simple syntactic analysis was chosen for processing efficiency, and because our domain-independent architecture avoids more knowledge intensive analysis.

The *constraints* of a rule can specify the entire phrase that matches the slot, the head of the phrase, or the head of each simple NP in an NPList slot. One type of constraint is an exact string constraint, such as the constraint head(NP1) = "cities" in the rule shown in Figure 2.3. Other constraints can specify that a phrase or its head must follow the orthographic pattern of a proper noun, or of a common noun. The rule *bindings* specify which slots or slot heads are extracted for each argument of the predicate. If the bindings have an NPList slot, a separate extraction is created for each simple NP in the list that satisfies all constraints. In the example rule, an extraction is created with the `City` argument bound to each simple NP in NPList2 that passes the proper noun constraint.

A final part of the rule is a list of *keywords* that is created from the context strings and any slots that have an exact word constraint. In our example rule, there is a single keyword phrase "cities such as" that is derived from slot NP1 and the immediately following context. A rule may have multiple keyword phrases if context or slots with exact string constraints are not immediately adjacent.

KnowItAll uses the keywords as search engine queries, then applies the rule to the Web page that is retrieved, after locating sentences on that page that contain the keywords. The rule language covers n-ary predicates with arbitrary relation name and multiple predicate arguments.

*Discriminators* The Extractor module uses extraction rules that apply to single Web pages and carry out shallow syntactic analysis. In contrast, the Assessor module uses discriminators that apply to search engine indices. These discriminators are analogous to simple extraction rules that ignore syntax, punctuation, capitalization, and even sentence breaks, limitations that are imposed by use of commercial search engine queries. On the other hand, discriminators are equivalent to applying an extraction pattern simultaneously to the entire set of Web pages indexed by the search engine.

A discriminator consists of an extraction pattern with alternating context strings and slots. There are no explicit or implicit constraints on the slots, and the pattern matches Web pages where the context strings and slots are immediately adjacent, ignoring punctuation, whitespace, or HTML tags. The discriminator for a unary predicate has a single slot, which we represent as an X here, for clarity of exposition. Discriminators for binary predicates have two slots, here represented as X and Y, for arguments 1 and 2 of the predicate, and so forth.

When a discriminator is used to validate a particular extraction, the extracted phrases are substituted into the slots of the discriminator to form a search query. This is described in more detail in Section 2.2.3. Figure 2.5 shows one of several possible discriminators that can be used for the predicate `City` and for the binary predicate `CeoOf(Person,Company)`.

We now describe how KnowItAll automatically creates a set of extraction rules and discriminator phrases for a predicate.

“the X scanner ” “ X is a scanner “ “ scanners such as X”

Figure 2.4: **Discriminator Examples.** Examples of discriminators used to assess candidate Scanner instances.

Discriminator for: City
“city X”
Discriminator for: CeoOf(Person,Company)
“X CEO of Y”

Figure 2.5: **Discriminator Examples.** When the discriminator for `City` is used to validate the extraction “Paris”, the Assessor finds hit counts for the search query phrase “city Paris”. Similarly, the discriminator for `CeoOf` validates Jeff Bezos as CEO of Amazon with the search query, “Jeff Bezos CEO of Amazon”.

### *2.2.2 Bootstrapping*

KnowItAll’s input is a set of *predicates* that represent classes or relationships of interest. The predicates supply symbolic names for each class (*e.g.* “MovieActor”), and also give one or more labels for each class (*e.g.* “actor” and “movie star”). These labels are the surface form in which a class may appear in an actual sentence. Bootstrapping uses the labels to instantiate extraction rules for the predicate from generic rule templates.

Figure 2.6 shows some examples of predicates for a geography domain and for a movies domain. Some of these are “unary” predicates, used to find instances of a class such as `City` and `Country`; some are “n-ary” predicates, such as the `capitalOf` relationship between `City` and `Country` and the `starsIn` relationship between `MovieActor` and `Film`. In this paper, we concentrate primarily on unary predicates and how KnowItAll uses them to extract instances of classes from the Web.

The first step of Bootstrapping uses a set of domain-independent generic extraction patterns (*e.g.* Figure 2.2). The pattern in Figure 2.2 can be summarized informally as `<class1> ‘‘such as’’ NPList` That is, given a sentence that contains the class label followed by “such as”, followed by a list of simple noun phrases, KnowItAll extracts the

Predicate: City
labels: “city”, “town”

Predicate: Country
labels: “country”, “nation”

Predicate: capitalOf(City,Country)
relation labels: “capital of”
class-1 labels: “city”, “town”
class-2 labels: “country”, “nation”

Predicate: Film
labels: “film”, “movie”

Predicate: MovieActor
labels: “actor”, “movie star”

Predicate: starsIn(MovieActor,Film)
relation labels: “stars in”, “star of”
class-1 labels: “actor”, “movie star”
class-2 labels: “film”, “movie”

Figure 2.6: **Example Predicates for a Geography Domain and for a Movies Domain.** The class labels and relation labels are used in creating extraction rules for the class from generic rule templates.

head of each noun phrase as a candidate member of the class, after testing that it is a proper noun.

Combining this template with the predicate `City` produces two instantiated rules, one for the class label “city” (shown in Figure 2.3) and a similar rule for the label “town”. The class-specific extraction patterns are:

“cities such as ” NPList

“towns such as ” NPList

Each instantiated extraction rule has a list of keywords that are sent as phrasal query terms to a search engine.

A sample of the syntactic patterns that underlie KnowItAll’s rule templates is shown in Figure 2.7.

Some of our rule templates are adapted from Marti Hearst’s hyponym patterns [48] and others were developed independently. The first eight patterns shown are for unary predicates whose pluralized English name (or “label”) matches <class1>. To instantiate the rules, the pluralized class label is automatically substituted for <class1>, producing patterns like “cities such as” NPList.

Bootstrapping also initializes the Assessor for each predicate in a fully automated manner. It first generates a set of discriminator phrases for the predicate based on class labels

```
NP "and other" <class1>
NP "or other" <class1>
<class1> "especially" NPList
<class1> "including" NPList
<class1> "such as" NPList
"such" <class1> "as" NPList
NP "is a" <class1>
NP "is the" <class1>

<class1> "is the" <relation> <class2>
<class1> "," <relation> <class2>
```

Figure 2.7: **Generic Unary and Binary Extraction Patterns.** The terms <class1> and <class2> stand for an NP in the rule pattern with a constraint binding the head of the phrase to a label of predicate argument 1 or 2. Similarly, <relation> stands for a phrase in the rule pattern with a constraint binding it to a relation label of a binary predicate.

and on keywords in the extraction rules for that predicate. Bootstrapping then uses the extraction rules to find a set of seed instances to train the discriminators for each predicate.

The Extractor matches the rule in Figure 2.3 to sentences in Web pages returned for the query. NP1 matches a simple noun phrase; it must be immediately followed by the string "such as"; following that must be a list of simple NPs. If the match is successful, the Extractor applies constraints from the rule. The head of NP1 must match the string "cities". The Extractor checks that the head of each NP in the list NPList2 has the capitalization pattern of a proper noun. Any NPs that do not pass this test are ignored. If all constraints are met, the Extractor creates one or more extractions: an instance of the class `City` for each proper noun in NPList2.

The rule in Figure 2.3 would extract three instances of `City` from the sentence "We service corporate and business clients in all major European cities such as London, Paris, and Berlin." If all the tests for proper nouns fail, nothing is extracted, as in the sentence "Detailed maps and information for several cities such as airport maps, city and downtown maps".

KnowItAll automatically formulates queries based on its extraction rules. Each rule has an associated search query composed of the rule's keywords. For example, if the pattern in Figure 2.3 was instantiated for the class `City`, it would lead KnowItAll to 1) issue the search-engine query "cities such as", 2) download in parallel all pages named in the engine's results, and 3) apply the Extractor to sentences on each downloaded page.

### *2.2.3 Assessor Description*

KnowItAll uses statistics computed by querying search engines to assess the likelihood that the Extractor's conjectures are correct. For example, the Assessor uses a form of *pointwise mutual information* (PMI) between words and phrases that is estimated from Web search engine hit counts in a manner similar to Turney's PMI-IR algorithm [114]. The Assessor computes the PMI-IR score for each extracted instance and multiple, *automatically generated discriminator phrases* associated with the class (such as "X is a city" for the class `City`).[1] For example, in order to estimate the likelihood that "Liege" is the name of a city, the Assessor might check to see if there is a high PMI-IR score between "Liege" and phrases such as "Liege is a city". More formally, let $I$ be an instance and $D$ be a discriminator phrase. We compute the PMI-IR score as follows:

$$\mathrm{PMI{-}IR}(I, D) = \frac{|\mathrm{Hits}(D + I)|}{|\mathrm{Hits}(I)|} \tag{2.1}$$

The PMI-IR score is the number of hits for a query that combines the discriminator and instance, divided by the hits for the instance alone.

The raw PMI score for an instance and a given discriminator phrase is typically a tiny fraction, perhaps as low as 1 in 100,000 even for positive instances of the class. This does not give the probability that the instance is a member of the class, only the probability of seeing the discriminator on Web pages containing the instance.

These mutual information statistics are treated as features that are input to a *Naive Bayes Classifier* (NBC) using the formula given in Equation 2.2. This is the probability that fact $\phi$ is correct, given features $f_1, f_2, \ldots f_n$, with an assumption of independence between the features.

[1]We use class names and the keywords of extraction rules to automatically generate these discriminator phrases.

$$P(\phi|f_1, f_2, \ldots f_n) = \frac{P(\phi)\prod_i P(f_i|\phi)}{P(\phi)\prod_i P(f_i|\phi) + P(\neg\phi)\prod_i P(f_i|\neg\phi)} \tag{2.2}$$

Our method to turn a PMI score into the conditional probabilities needed for Equation 2.2 is straightforward. The Assessor takes a set of $k$ positive and $k$ negative seeds for each class and finds a threshold on PMI scores that splits the positive and negative seeds. It then uses a tuning set of another $k$ positive and $k$ negative seeds to estimate $P(PMI > thresh|class)$, $P(PMI > thresh|\neg class)$, $P(PMI \leq thresh|class)$, and $P(PMI \leq thresh|\neg class)$, by counting the positive and negative seeds (plus a smoothing term) that are above or below the threshold. We used $k = 10$ and a smoothing term of 1 in the experiments reported here.

In a standard NBC, if a candidate fact is more likely to be true than false, it is classified as true. However, since we wish to be able to trade precision against recall, we record the crude probability estimates computed by the NBC for each extracted fact. By raising the probability threshold required for a fact to be deemed true, we increase precision and decrease recall; lowering the threshold has the opposite effect. We found that, despite its limitations, NBC gave better probability estimates than the logistic regression and Gaussian models we tried. Several open questions remain about the use of PMI for information extraction. Even with the entire Web as a text corpus, the problem of sparse data remains. The most precise discriminators tend to have low PMI scores for numerous positive instances, often as low as $10^{-5}$ or $10^{-6}$. This is not a problem for prominent instances that have several million hits on the Web. If an instance is found on only a few thousand Web pages, the expected number of hits for a positive instance will be less than 1 for such a discriminator. This leads to false negatives for the more obscure positive instances.

A different problem with using PMI is homonyms — words that have the same spelling, but different meanings. For example, Georgia refers to both a state and country, Normal refers to a city in Illinois and a socially acceptable condition, and Amazon is both a rain forest and an on-line shopping destination.

Another issue is in the choice of a Naive Bayes Classifier. Since the Naive Bayes Classifier is notorious for producing polarized probability estimates that are close to zero or to one, the estimated probabilities are often inaccurate. However, as [30] points out, the classifier is surprisingly effective because it only needs to make an ordinal judgment (which class is more likely) to classify instances correctly. Similarly, our formula produces a reasonable *ordering* on the likelihood of extracted facts for a given class. This ordering is sufficient for KnowItAll to implement the desired precision/recall tradeoff.

## *2.3 Related Work*

One of KnowItAll's main contributions is adapting Turney's PMI-IR algorithm [114, 115, 117] to serve as validation for information extraction. PMI-IR uses search engine hit counts to compute pointwise mutual information that measures the degree of correlation between a pair of words. Turney used PMI from hit counts to select among candidate synonyms of a word, and to detect the semantic orientation of a phrase by comparing its PMI with positive words (*e.g.* "excellent") and with negative words (*e.g.* "poor"). Other researchers have also made use of PMI from hit counts: [64] validates proposed question-answer pairs for a QA system by learning "validation patterns" that look for the contexts in which the proposed question and answer occur in proximity; Uryupina [119] classifies proposed instances of geographical classes by embedding the instance in discriminator phrases much like KnowItAll's, which are then given as features to the Ripper classifier.

KnowItAll is distinguished from many Information Extraction (IE) systems by its novel approach to bootstrap learning, which obviates hand-labeled training examples. Unlike IE systems that use supervised learning techniques such as *hidden Markov models* (HMMs) [39], rule learning [105, 16, 21], maximum entropy [78], or Conditional Random Fields [70], KnowItAll does not require any manually-tagged training data.

Bootstrap learning is an iterative approach that alternates between learning rules from a set of instances, and finding instances from a set of rules. This is closely related to co-training [11], which alternately learns using two orthogonal view of the data. IE systems that use bootstrapping include [96, 2, 13, 26, 24, 22]. These systems begin with a set of hand-tagged seed instances, then alternately learn rules from seeds, and further seeds from rules. KnowItAll is unique in not requiring hand-tagged seeds, but instead begins with a domain-independent set of *generic extraction patterns* from which it induces a set of seed instances. KnowItAll's use of PMI validation helps overcomes the problem of maintaining high precision, which has plagued previous bootstrap IE systems.

KnowItAll is able to use weaker input than previous IE systems because it relies on the scale and redundancy of the Web for an ample supply of simple sentences. This notion of *redundancy-based extraction* was introduced in Mulder [58] and further articulated in AskMSR [69]. Of course, many previous IE systems have extracted more complex relational information than KnowItAll. KnowItAll is effective in extracting $n$-ary relations from the Web, but we have yet to demonstrate this experimentally.

Several previous projects have automated the collection of information from the Web with some success. Information extraction systems such as Google's Froogle, Whizbang's Flipdog, and Elion, collected large bodies of facts but only in carefully circumscribed domains (*e.g.*, job postings), and only after extensive domain-specific hand tuning. KnowItAll is both highly automated and domain independent. In fairness, though, KnowItAll's redundancy-based extraction task is easier than Froogle and Flipdog's task of extracting "rare" facts each of which only appears on a single Web page. Semantic tagging systems, notably SemTag [28], perform a task that is complementary to that of KnowItAll. SemTag starts with the TAP knowledge base and computes semantic tags for a large number of Web pages. KnowItAll's task is to automatically extract the knowledge that SemTag takes as input.

KnowItAll was inspired, in part, by the WebKB project [25]. However, the two projects rely on very different architectures and learning techniques. For example, WebKB relies on supervised learning methods that take as input hand-labeled hypertext regions to classify Web pages, whereas KnowItAll employs unsupervised learning methods that extract facts by using search engines to home in on easy-to-understand sentences scattered throughout the Web. Finally, KnowItAll also shares the motivation of Schubert's project [99], which seeks to derive general world knowledge from texts. However, Schubert and his colleagues have focused on highly-structured texts such as WordNet and the Brown corpus whereas KnowItAll has focused on the Web.

# Chapter 3
# CLASS EXTRACTION FROM THE WEB

## *3.1 Introduction*

Many researchers have used open-domain or domain-specific text corpora in order to automatically build or augment *term lexicons* and *concept hierarchies* used by a variety of applications (*e.g.*, question answering, information extraction, information retrieval, etc.). Recently, the large amount of textual Web data has become an attractive alternative to newspaper corpora and previously used domain specific texts.

This chapter describes our work on high precision, Web-based extraction of subclasses of given general classes and new relevant concepts for a specified domain. High precision subclass extraction can be used to automatically augment WordNet's IS-A hierarchy from the Web; the chapter includes a discussion of promising preliminary results for this task.

## ***3.2 Subclass Extraction***

This section describes how the basic extraction model introduced in Chapter 1 can be instantiated to accommodate *high precision subclass extraction (SE)* and introduces two methods for increasing the coverage of the subclass extraction module. Finally, it reports on the significant impact of subclass extraction on the yield of the basic KnowItAll system [35].

Subclass extraction is achieved by recursively applying the main loop of the previously introduced Web-based extraction model. In the following we discuss the various flavors of subclasses and subclass extraction, present the basic subclass extraction method ($SE_{base}$), introduce two variations aimed at increasing SE's recall ($SE_{self}$ and $SE_{iter}$) and finally describe encouraging results for a number of different classes.

### *3.2.1 Subclass Types*

We distinguish between finding subclasses in a *context-independent* manner versus finding subclasses in a *context-dependent* manner. The term *context* refers to a set of keywords provided by the user that suggest a knowledge domain of interest (e.g., the pharmaceutical domain, the political domain, etc.). In the absence of a domain description, our system finds subclasses in a context-independent manner and they can differ from context-dependent subclasses. For instance, if we are looking for any subclasses of Person (or People), Priest would be a good candidate. However, if we are looking for subclasses of Person (or People) in a Pharmaceutical context, Priest is probably not a good candidate, whereas Pharmacist is. We also distinguish between *named subclasses* and *derived subclasses*. *Named subclasses* are represented by novel terms, whereas *derived subclasses* are phrases whose head noun is the same with the name of the superclass. For instance, *Capital* is a named subclass of *City*, whereas *European City* is a derived subclass of *City*.

### *3.2.2 Extracting Candidate Subclasses*

The main components of the subclass extraction system are the Extractor and the Assessor. The $SE_{base}$ Extractor is an instantiation of the Extractor module introduced in Chapter 1.

Table 3.1: **Extraction Rule Patterns for Subclass Extraction**. Notation: $Class_1$ = target class, $NP$ = noun phrase, $Subclass_1$,$Subclass_2$ = subclasses of $Class_1$. Note that the last two rules can only be used once subclasses of the target class have already been found.

| **Pattern** | **Constraints** |
|---|---|
| $Class_1$ {"," } "such as" $NP$ | commonNoun(head($NP$)) |
| "such" $Class_1$ "as" $NP$ | commonNoun(head($NP$)) |
| $NP$ {"," } "and other" $Class_1$ | commonNoun(head($NP$)) |
| $NP$ {"," } "or other" $Class_1$ | commonNoun(head($NP$)) |
| $Class_1$ {"," } "including" $NP$ | commonNoun(head($NP$)) |
| $Class_1$ {"," } "especially" $NP$ | commonNoun(head($NP$)) |
| $Subclass_1$ "and" $NP$ | commonNoun(head($NP$)) |
| $Subclass_1$ {","} $Subclass_2$ {"," } "and" $NP$ | commonNoun(head($NP$)) |

Its input consists of domain-independent extraction rules for generating candidate terms, for which matches are found on the Web. The generic rules that extract instances of a class will also extract subclasses, with some modifications. To begin with, the rules need to distinguish between instances and subclasses of a class. Rules for instances already contain a proper noun test (using a part-of-speech tagger and a capitalization test). Rules for extracting subclasses instead check that the head noun of the extracted noun phrases is a common noun (i.e., not capitalized). While these tests are heuristic, they work reasonably well in practice and the Assessor weeds out most erroneous extractions.

The patterns for our subclass extraction rules appear in Table 3.1. Most of our patterns were simple variations of well-known ones in the information-extraction literature [48]. Note that the last two rules can only be used once two subclasses of the class have already been found. Also, when we perform subclass extraction in a given context, the search engine queries contain a relevant keyword together with the instantiated extraction rule (for instance, "pharmaceutical" in the case of the Pharmaceutical domain).

### *3.2.3 Assessing Candidate Subclasses*

The $SE_{base}$ Assessor decides which of the candidate subclasses from the $SE_{base}$ Extractor are correct. First, the Assessor checks the morphology of the candidate term, since some subclass names are formed by attaching a prefix to the name of the class (e.g., "microbiologist" is a subclass of "biologist"). Then, if the subclass extraction is done in a

Table 3.2: **Subclass Extraction results for the Scientist and Film classes.** Notation: Total is the number of *correct* subclasses, NotWordNet is the proportion of the correctly identified subclasses missing from WordNet.

| Method | **Scientist** | | | | **Film** | | | |
|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | NotWordNet | Total | Precision | Recall | NotWordNet | Total |
| $SE_{base}$ | 0.91 | 0.28 | 0.08 | 11 | 1.0 | 0.36 | 0.5 | 8 |
| $SE_{self}$ | 0.87 | 0.69 | 0.15 | 27 | 0.94 | 0.77 | 0.82 | **17** |
| $SE_{iter}$ | 0.84 | 0.74 | 0.17 | **29** | 0.93 | 0.68 | 0.8 | 16 |

Table 3.3: **Subclass Extraction results for People, Product and Organization classes in the context of the Pharmaceutical domain.** Notation: P = precision, R = recall, Total = the number of *correct* subclasses, NW = the proportion of the correctly identified subclasses missing from WordNet.

| Method | **People** | | | | **Organization** | | | | **Product** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | R | NW | Total | P | R | NW | Total | P | R | NW | Total |
| $SE_{base}$ | 1.0 | 0.28 | 0.0 | 14 | 0.92 | 0.20 | 0.09 | 11 | 0.88 | 0.44 | 1.0 | 31 |
| $SE_{self}$ | 1.0 | 0.86 | 0.0 | 42 | 0.87 | 0.84 | 0.36 | 47 | 0.86 | 0.74 | 1.0 | 51 |
| $SE_{iter}$ | 0.95 | 0.94 | 0.0 | **46** | 0.89 | 0.95 | 0.22 | **52** | 0.84 | 0.88 | 1.0 | **62** |

context-independent manner, the Assessor checks whether a subclass is a hyponym of the class in WordNet and if so, it assigns it a very high probability.

The rest of the extractions are evaluated in a manner similar to the assessment strategy described in Chapter 1. The Assessor computes co-occurrence statistics of candidate terms with a set of class discriminators. Such statistics represent features combined using a naive Bayesian probability update [33]. The $SE_{base}$ Assessor uses a training set in order to estimate the conditional probabilities of observing a feature if an example is or is not a correct subclass. The Assessor ranks the set of proposed subclasses based on the number of matching extraction rules and then re-ranks the top $n$ ($n = 40$) according to their average PMI score with respect to the set of discriminators for the class. The top $k = 10$ subclass candidates that also co-occur with every discriminator in the set represent the set of positive examples for the class. The negative examples for each class are collected from among the positive examples for the other classes in the ontology.

### *3.2.4 Experimental Results: Basic Subclass Extraction*

Before we discuss our experimental results we briefly refer back to the distinction between *named* and *derived* subclasses. Although both types of subclass terms can be extracted in our framework, we focused on the *named* subclasses, as our initial goal was to find new terms in order to improve the yield of the KnowItAll system (KnowItAll extraction rules that use derived subclasses tend to extract many of the same instances as the rules using the name of the target class). Later in the chapter we will turn our attention to extracting *derived* subclasses.

We have evaluated our basic subclass extraction method in two different settings.

a) **Context-independent SE** First, we chose three classes, Scientist, City and Film and looked for context-independent subclasses using the $SE_{base}$ approach described above. $SE_{base}$ found only one named subclass for City, "capital", which is also the only one listed in the WordNet hyponym hierarchy for this class. $SE_{base}$ found 8 correct subclasses for Film and 11 for Scientist - this confirmed our intuition that subclass extraction would be most successful on general classes, such as Scientist and least successful on specific classes such as City. As shown in Table 3.2, we have evaluated the output of $SE_{base}$ along four metrics: precision, recall, total number of correct subclassses and proportion of (correct) found subclasses that do not appear in WordNet. As we can see, $SE_{base}$ has high-precision but relatively low recall, reflecting the low recall of our domain-independent patterns.

b) **Context-dependent SE** A second evaluation of $SE_{base}$ (detailed in Table 3.3 ) was done for a context-dependent subclass extraction task, using as input three categories that were shown to be productive in previous semantic lexicon acquisition work [88]: People, Products and Organizations in the Pharmaceutical domain. $SE_{base}$ exhibits the same high-precision/low-recall behavior we noticed in the context-independent case. We also notice that most of the subclasses of People and Organizations are in fact in WordNet, whereas none of the found subclasses for Products in the Pharmaceutical domain appears in WordNet.

Next, we investigate two methods for increasing the recall of the subclass extraction module.

## *3.3 Improving Subclass Extraction Recall*

Generic extraction rules have low recall and do not generate all of the subclasses we would expect. In order to improve our subclass recall, we add another extraction-and-verification step. After a set of subclasses for the given class is obtained in the manner of $SE_{base}$, the high-recall enumeration rules in Table 3.1 are seeded with known subclasses and extract additional subclass candidates. For instance, given the sentence "Biologists, physicists and chemists have convened at this inter-disciplinary conference.", such rules identify "chemists" as a possible sibling of "biologists" and "physicists". The candidate subclass sets extracted in this fashion contain reliable seed subclasses (whose probability was already determined by the Naive Bayes Assessor), terms previously classified as negative examples and novel terms. We experiment with two methods, $SE_{self}$ and $SE_{iter}$ in order to assess the extractions obtained at this step.

a) $SE_{self}$ is a simple assessment method based on the empirical observation that an extraction matching a large number of different enumeration rules is likely to be a good subclass candidate. We have tried to use the enumeration rules directly as features for a Naive Bayes classifier, but the very nature of the enumeration rule instantiations ensures that positive examples don't have to occur in any specific instantiation, as long they occur frequently enough. We simply convert the number of different enumeration rules matched by each example and the average number of times an example matches its corresponding rules into boolean features (using a learned threshold). Since we have a large quantity of unlabeled data at our disposal, we estimate the thresholds and train a simple Naive-Bayes classifier using the *self-training* paradigm [79], chosen as it has been shown to outperform EM in a variety of situations. At each iteration, we label the unlabeled data and retain the example labeled with highest confidence as part of the training set. The procedure is repeated until all the unlabeled data is exhausted. The extractions whose probabilities are greater than 0.8 represent the final set of subclasses (since subclasses are generally used by KnowItAll for instance extraction, bad subclasses translate into time wasted by the system and as such, we retain only candidate subclasses whose probability is relatively high).

b) $SE_{iter}$ is a heuristic assessment method that seeks to adjust the probabilities assigned to the extractions based on *confidence scores* assigned to the enumeration rules in a recursive fashion. The *confidence score* of a rule is given by the average probability of extractions matched by that rule. After rule confidence scores have been determined, the extraction matching the most rules is assigned a probability $p = \frac{c(R1)+c(R2)}{2}$ , where $R1$ and $R2$ are the two matching rules with highest confidence scores. The rule confidence scores are then re-evaluated and the process ends when all extractions have been assigned a probability. This scheme has the effect of clustering the extractions based on the rules they match and it works to the advantage of good subclasses that match a small set of good extraction rules. However, as we will later see, this method it is sensitive to noise. As in the case of $SE_{self}$, we only retain the extractions whose probability is greater than 0.8.

### *3.3.1 Experimental Results: Improved Subclass Extraction*

We evaluated the methods introduced above on two of the three context-independent classes (Scientist and Film) in Table 3.3.[1] We also evaluated the methods on all three Pharmaceutical domain classes (People, Product, Organization) in Table 3.2. We found that both $SE_{self}$ and $SE_{iter}$ significantly improved upon the recall of the baseline method: for both, this increase in recall is traded for a loss in precision. $SE_{iter}$ has the highest recall, at the price of an average 2.3% precision loss with respect to $SE_{base}$. In the future, we will perform additional experiments to assess which one of the two methods is less sensitive to noise, but based upon inspection of the test set and the behavior of both methods, $SE_{self}$ appears more robust to noise than $SE_{iter}$.

Table 3.4 contains some examples of extracted Scientist subclasses.

Another potential benefit of subclass extraction is an increase in the number of class instances that KnowItAll is able to extract from the Web. In the case of the Scientist class, for example, the number of scientists extracted by KnowItAll at precision 0.9 increased by a factor of 10. $SE_{iter}$ was used to extract subclasses and add them to the ontology. We do not see this benefit for classes such as City for which we are only able to extract a couple of named subclasses (e.g., "capital").

[1]We didn't have enough subclasses to instantiate enumeration patterns for City as $SE_{base}$ only identified one named City subclass.

Table 3.4: **Sample Named Subclasses of the Scientist Target Class.** The italics indicate subclass terms missing from WordNet.

| |
|---|
| biologist zoologist astronomer meteorologist |
| mathematician economist geologist sociologist |
| chemist oceanographer anthropologist pharmacist |
| psychologist climatologist paleontologist *neuropsychologist* |
| *engineer* microbiologist |

### *3.3.2 Discussion*

It is somewhat surprising that simple features such as the number of rules matching a given extraction are such good predictors of a candidate representing a subclass. We attribute this to the redundancy of Web data (we were able to find matches for a large number of our instantiated candidate rules) and to the semantics of the enumeration patterns. The subclass sets from $SE_{self}$ and $SE_{iter}$ contain many of the same candidates, although $SE_{iter}$ typically picks up a few more.

Another interesting observation is that the different sets of extracted subclasses have widely varying degrees of overlap with the hyponym information available in WordNet. For example, all subclasses identified for People are in WordNet whereas none of those Products appear there (e.g., “antibiotic”, “antihistamine”, etc.).

## *3.4 Related Class Extraction*

In the related class extraction task, our system starts with a set of predicates (and optionally, a set of instances and/or keywords) relevant to a specific topic and then automatically identifies and explores new *related concepts*, that is, concepts pertaining to the same topic. We refer to the initial set of predicates/instances/keywords as the system's *information focus* and the task is identifying other predicates relevant to the current focus.

For instance, in the Computer domain, the information focus might contain four classes: `Computer`, `Chip`, `Monitor` and `Disk`, together with a few instances of each. We would like our system to automatically identify `Modem`, `Drive` and other such conceptually related classes. The challenge is to remain in focus; for instance, in the case of the Geography domain it is easy enough to drift into a related area, such as Economy or Politics.

The *RCE (related class extraction)* module proceeds in an iterative fashion: at each iteration, it uses a sample of the known domain-specific classes to instantiate a set of extraction patterns and then assesses the relevance of the extractions obtained in this manner with respect to the information focus. The iterative procedure ends after a set number of iterations or when its signal-to-noise ratio drops below a set threshold.

Related class extraction can be achieved by instantiating the basic extraction model introduced in Chapter 1 in a manner similar to that used to solve the subclass extraction problem. In the following, we describe this process in more detail.

### *3.4.1 Extracting Candidate Classes*

The RCE Extractor has the same design as the basic Extractor module in Chapter 1. Unlike the rules for subclass extraction, the rules for related class extraction also make use of instance information. For example, given the class City and a few seed instances (Paris, London, etc.), we can find terms potentially related to City using rules instantiated with either the name of the class ("city and other $C$ ", where $C$ is a candidate term) or the name of a seed instance ("Paris and other $C$", where $C$ is a candidate term).

Table 3.5: **Related Class Extraction results in the Geography and Computer domains**. We report on the precision and recall metrics for each method, together with the number of relevant terms extracted.

| Method | **Geography** | | | **Computer** | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | Total | Precision | Recall | Total |
| $RCE_{base}$ | 0.74 | 0.48 | 25 | 0.93 | 0.65 | 53 |
| $RCE_{self}$ | 0.76 | 0.82 | 43 | 0.91 | 0.81 | 66 |
| $RCE_{iter}$ | 0.71 | 0.86 | **45** | 0.88 | 0.84 | **69** |

### *3.4.2 Assessing Candidate Classes*

The RCE Assessor takes as input the set of candidate concepts found in the previous step and computes the *relevance* of each class name with respect to the given information focus. The *relevance* of a new class name is measured on the basis of web-scale PMI statistics of the candidate name with information focus terms. The relevance measure attempts to *cheaply* eliminate candidate terms that are not domain-specific (more advanced relevance measures, based on identifying probable relationships between a candidate term and known domain-specific terms, are also more expensive).

Given a domain specific term $T$ and a candidate class $C$, we compute $C$'s semantic similarity to $T$ using the PMI score below:

$$\text{PMI}-\text{IR(C, T)} = \frac{|Hits(C,T)|}{|Hits(C)| * |Hits(T)|}.$$

Given a set of terms $T_s$ describing a domain-specific class $T$ (class name, names of class instances) and a new candidate class $C$, we first compute the similarity between $C$ and each term describing $T$; we then average the resulting scores to get a measure of the similarity between $C$ and $T_s$:

$$PMI_{avg}(C,T_s) = \frac{\sum_{e \in T_s} \text{PMI}-\text{IR(C, e)}}{|T_s|}$$

where $e$ is some element in $T_s$.

$PMI_{avg}(C,T_s)$ measures the average relevance of $C$ to the core class $T$ using all the available information about $T$. These final scores are used to determine whether a given

Table 3.6: **Sample Classes Discovered by the Baseline RCE Method ($RCE_{base}$) in the Computer Domain.**

| |
|---|
| keyboard drive modem mouse circuit |
| hardware battery laptop memory microprocessor |
| peripheral semiconductor bus server |

class $C$ is relevant or not to the given information focus. We use a set of learned thresholds in order to convert the relevance scores into boolean features. The features are then combined using a Naive Bayes classifier in order to predict whether the candidate name is relevant with respect to the given information focus. The Assessor considers all candidates whose relevance probability is greater than 0.8 to be relevant - we refer to this basic RCE extraction and assessment procedure as $RCE_{base}$.

### *3.4.3 Increasing Related Class Extraction Recall*

In order to increase the recall of the basic RCE module, we experimented with two modified RCE methods, $RCE_{self}$ and $RCE_{iter}$, with the same structure as the ones used to increase the recall of the subclass extraction component. These are versions of $RCE_{base}$ in which the set of good extractions computed at the end of each iteration in the manner described above is augmented using enumeration rules. Once new extractions are obtained in this manner, they are evaluated with respect to the enumeration rules that extracted them (as described in the subclass extraction section). They are not evaluated with respect to the information focus since this is expensive in terms of search engine queries and a lot of good class names do not co-occur enough times with information focus terms.

### *3.4.4 Experimental Results*

We tested our methods for extracting classes relevant to a given domain description on two domains: Geography and Computers. All RCE methods started with basic initial knowledge of each domain in the form of four seed classes: `Cities`, `States`, `Countries`, and `Rivers` for the Geography domain, and `Computers`, `Chips`, `Monitors`, and `Disks` for the Computer domain. Due to time constraints, we first performed two iterations of the $RCE_{base}$ method in order to obtain a non-trivial set of good extractions (18 for the Geography domain and 32 for the Computer domain). We then performed a third iteration, this time comparing the simple $RCE_{base}$ method with its modified versions, $RCE_{self}$ and $RCE_{iter}$. Table 3.5 reflects the results after the third iteration and Table 3.6 shows some examples of newly identified classes for the Computer domain. We report on the precision and recall metrics for our three methods— their performance was evaluated by hand-labeling the extracted terms. We also include the total number of relevant classes extracted by each method.

As we had hoped, $RCE_{self}$ and $RCE_{iter}$ led to an increase in RCE recall. We noticed that the $RCE_{iter}$ led to the highest increase in recall, but in the context of a precision drop of 3.5% on average, while $RCE_{self}$ managed to increase the recall, while maintaining (and in the case of the Geography domain, increasing) precision. In the computer domain, where we had started with a large seed set, a lot of the extractions had already been found by our initial set of domain-independent patterns and as such, the recall increase was not as dramatic as in the Geography domain, where the number of relevant new terms almost doubled for $RCE_{iter}$. Overall, the only statistically significant differences are the jumps in recall from $RCE_{base}$ to the more sophisticated $RCE_{self}$ and $RCE_{iter}$.

In future work, we plan to analyze the behavior of these methods as we execute longer runs for all of them. It will also be interesting to see how the description of the information focus should adjust over time due to the exhaustion of immediately relevant terms. We want to broadly explore a given domain but not drift into a different domain without realizing it.

## *3.5 Related Work*

There has been previous NLP work focused on acquiring domain-specific lexicons from corpora. The bulk of this work consists of weakly supervised learning algorithms for acquiring members of one or several semantic categories using a bootstrapping approach in conjunction with lexical co-occurrence statistics, specific syntactic structures [98], generic lexico-syntactic patterns [48] and detailed extraction patterns, designed to capture role relationships [88]. More recently, researchers exploring biomedical literature [93] have looked at identifying concepts related to a set of target concepts using a mixture of data mining techniques and natural language processing techniques.

Our approach is Web-based instead of corpus-based and so we use search engines to compute co-occurrence statistics and take advantage of the large number of instantiated enumeration patterns available on the Web.

The ontology development field has used exhaustive conceptual clustering techniques to acquire taxonomic relationships (more specifically, the IS-A relationship) from text. Typically an initial taxonomy is built using a combination of statistical and lexical information and the taxonomy is then refined (using automatic classification methods) as new concepts are identified in the text [63]. Our subclass acquisition module uses different types of information (Web-scale co-occurrence statistics, a large number of enumeration extraction rules) and it evaluates the acquired information using a different methodology.

The subclass extraction approach closest to ours is described in [106]. The authors use a single lexical pattern (search engine queries) that yield sentences containing potential hypernyms/hyponyms for a given term. After detecting synonymous words using WordNet, the method uses frequency information to identify the most likely hypernym/hyponym candidates. The paper does not give a precision figure, just examples of extractions, and reports on tests in just one domain (the computer domain).

## 3.6 Augmenting WordNet Using Web Data

WordNet [73] is a widely used lexical resource with widely recognized limitations such as the general lack of domain-specific terms and the limited coverage of useful relations (*e.g.*, Cause, Part-of, etc.). This section describes our 2004 proposal [89] for a set of large-scale WordNet extensions made possible by our Web-based information extraction architecture.

The contributions of this section are as follows:

1) We describe how the Web-based information extraction architecture at the core of this thesis and especially the subclass extraction module previously described in this chapter can be used to augment the WordNet IS-A hierarchy using Web data.

2) We describe two assessment methods, *baseline assessment* and *collective assessment*, which are used to evaluate candidate instances of the IS-A relationship. We show that *collective assessment* improves upon the performance of the *baseline assessment* method.

3) We describe a set of preliminary experiments and report on estimates of how long it would take our system to complete a large scale WordNet extension.

The rest of this chapter is organized as follows: we start with an overview of our system, we then describe in detail how candidate IS-A links are generated and validated using Web data, we report on a set of preliminary experiments and give our estimates of how long it would take to complete a large scale WordNet extension; finally, we discuss relevant related work.

### 3.6.1 Proposed Extensions and System Overview

In this section we give an overview of the proposed WordNet extensions and briefly describe the steps of our solution (see Figure 3.1).

We focus on the following two types of extensions (see Table 3.7 for examples):

a) augmenting WordNet's IS-A hierarchy with links between *known* concepts and *novel* concepts: *e.g.*, given the concept $segment_1$, we identify descendants such as $marketSegment_0$.

b) augmenting WordNet's IS-A hierarchy with missing links among *known* concepts: *e.g.*, given the concept $domesticAnimal_0$, we identify corresponding descendants such as $cow_0$.

Table 3.7: **Examples of IS-A Links Automatically Obtained from Web Data (additional examples can be see in Appendix A.1).**

| **Type of Extracted IS-A Link** | **Concept Pair** |
|---|---|
| Missing IS-A Link Between Known Concepts | $(herbivore_0,\ equine_0)$ |
| | $(domesticAnimal_0,\ cow_0)$ |
| | $(endoparasite_0,\ tapeworm_0)$ |
| | $(endoparasite_0,\ hookworm_0)$ |
| Known Concept $\leftarrow$ Novel Concept Link | $(segment_1,\ marketSegment_0)$ |
| | $(animal_0,\ forestAnimal_0)$ |
| | $(construction_4,\ tunneling_0)$ |
| | $(construction_4,\ paving_0)$ |
| | $(psychologist_0,\ neuropsychologist_0)$ |
| | $(endoparasite_0,\ whipworm_0)$ |

Our system starts with a seed set of target WordNet concepts - in the following we describe how such concepts are chosen and then describe the WordNet extension process.

*Seed WordNet Concepts*

Based on a brief initial analysis, we imposed the following conditions on a seed target concept:

a) The concept must have $k$ or fewer descendants (not including named instances) - in our experiments, we used the experimentally set value $k = 10$. Focusing on concepts with few descendants (*e.g.*, $domesticAnimal_0$ [2]) is more likely to result in the addition of new IS-A links.

b) The first term in the concept's synset must have $k'$ or less senses (in our experiments we used $k' = 4$). The proliferation of senses for some words in WordNet is a well documented problem - we noticed that in many cases the same word is used to name two different concepts that are highly related (even subclasses of the same parent concept). The WordNet project statistics show that on average WordNet terms have 2 senses, which means that this condition does not significantly reduce the number of potential candidate target concepts

[2]The examples in this chapter are based on the 2004 version of WordNet - the team behind WordNet has recently released a more complete version in which some of the classes listed in this chapter have more complete hyponym lists.

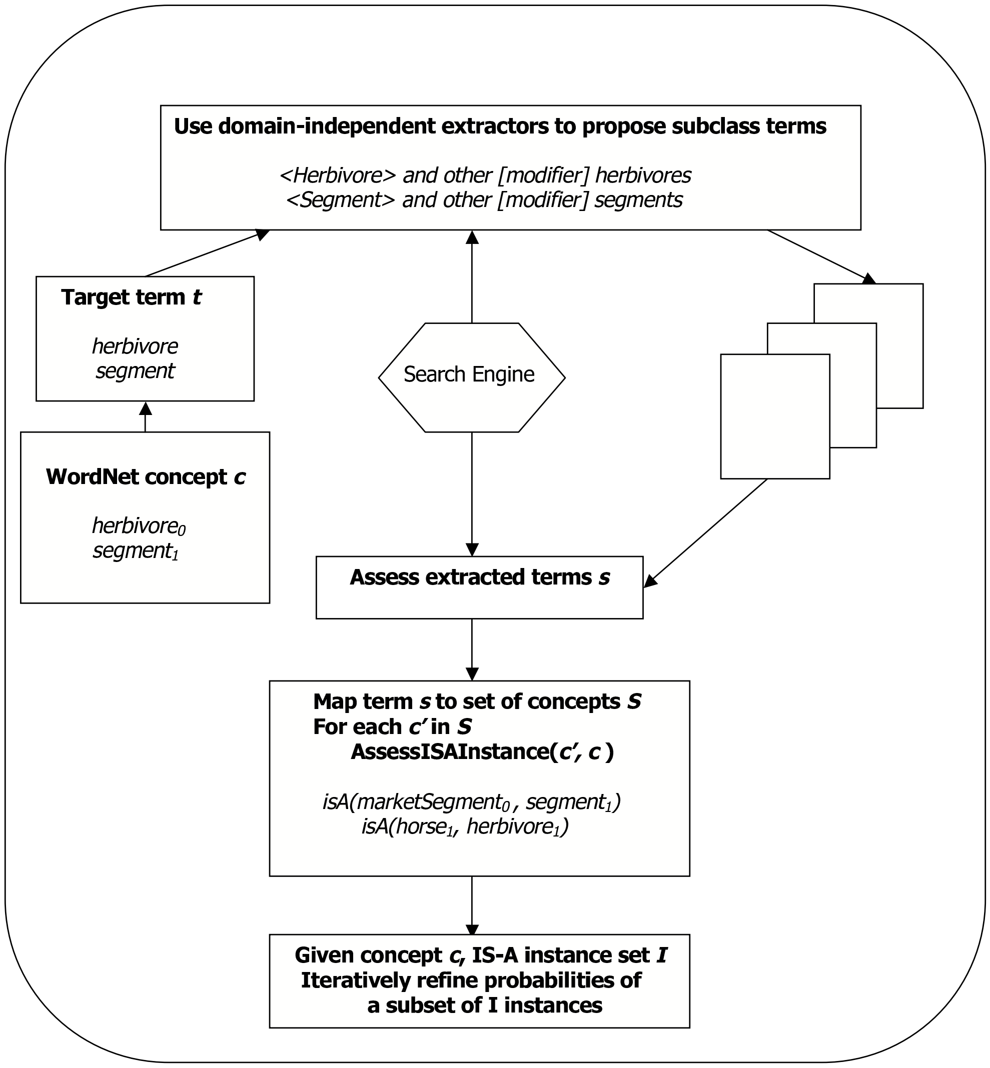

Use domain-independent extractors to propose subclass terms
<Herbivore> and other [modifier] herbivores
<Segment> and other [modifier] segments
Target term t
herbivore
segment
Search Engine
WordNet concept c
herbivore0
segment1
Assess extracted terms s
Map term s to set of concepts S
For each c′ in S
AssessISAInstance(c′, c )
isA(marketSegment0 , segment1)
isA(horse1, herbivore1)
Given concept c, IS-A instance set I
Iteratively refine probabilities of
a subset of I instances

Figure 3.1: **Augmenting WordNet's IS-A Hierarchy Using Web Data**. Given a target concept, its most common name is used to extract and assess *subclass terms*. The extracted subclass terms are then used to propose concept-level IS-A links that are subsequently assessed. The WordNet IS-A hierarchy is augmented with links between known and novel concepts as well as missing links among known concepts.

but merely ensures that we eliminate a source of ambiguity.

c) When a concept that satisfies the above conditions is added to the seed set, its descendants are added as well (with the exception of those not meeting condition b)).

In the following we describe how novel IS-A links are identified from Web data starting with the set of seed concepts.

*System Overview*

Given a target concept (*e.g.*, $herbivore_0$), the system chooses the first synonym in the corresponding synonym set as the *target term* (e.g., `herbivore`). The choice is motivated by the fact that the terms in a WordNet synset are ordered by frequency in use, which means that the first term represents the most commonly used name for the current concept.

An overview of the WordNet extension process can be seen in Figure 3.1. The system starts with a concept seed set: for each concept, it extracts and assesses a set of *subclass terms*. Each extracted subclass term is mapped to a set of WordNet concepts as follows: the novel terms are mapped to newly created WordNet concepts and the terms already in the WordNet dictionary are mapped to the corresponding sets of WordNet senses. The system then establishes the set of IS-A links between the seed WordNet concepts and the set of WordNet concepts corresponding to mined subclass terms. This step includes creating IS-A links between seed concepts and novel concepts as well as augmenting the IS-A hierarchy with missing links among existent WordNet concepts. Our system uses two assessment methods for concept-level IS-A links: *baseline assessment* and *dependency-based collective updates*.

In the following, we describe this process in more detail.

## *3.6.2 Extracting and Assessing Subclass Terms*

Given a target term, the system uses a version of the *subclass extraction* module described earlier in this chapter in order to extract candidate subclass terms in the form of *simple* noun phrases (*e.g.*, "physicist") and two types of *complex* noun phrases: compound nouns (*e.g.*, "car service") and complex noun phrases containing *objective* adjectival modifiers (*e.g.*, "urban area", "electric car"). The system discards noun phrases with subjective adjectival modifiers (*e.g.*, "good scientist") - for this purpose we rely on a previously compiled list of modifiers: starting with a seed set of subjective words (*e.g.*, "good", "bad", etc.), we iteratively expand the set using synonym and antonym information culled from WordNet. This methodology has been used in many NLP projects - an alternative, but much more expensive way to distinguish between subjective and objective modifiers is to use Turney's Web-based algorithm (as described in Chapter 5).

The candidate terms are then assessed using a slightly modified version of the *subclass assessment* module described earlier in this chapter (see subsection 3.6.5 for a brief discussion of these modifications). The assessed terms with corresponding probabilities $p \geq 0.8$ are used to propose novel IS-A links as described in the following.

### *3.6.3 Generating and Assessing Concept-Level IS-A Links: Baseline Assessment*

Each novel subclass term is mapped to a newly created WordNet concept (*e.g.*, "car service" is mapped to $carService_0$). Each subclass term already in the WordNet dictionary is mapped to the corresponding set of WordNet synsets (*e.g.*, "horse" is mapped to $\{horse_0, horse_1\}$).

Given a seed concept $c$ and a mined subclass term $t$, let $c'$ denote one of the concepts to which $t$ is automatically mapped. For all such $c$ and $c'$, the system assesses all potential relationship instances $isA(c', c)$ (*e.g.*, $isA(horse_0, domesticAnimal_0)$). In the following, we describe the first of the two assessment methods used by our system, *baseline assessment*.

Given a WordNet concept $c$, we characterize it using the following *descriptions*:

- Let $d(c)$ denote a set of terms that contains the first (and if it exists, the second) synonym in $c$'s WordNet synset. For new concepts, $d(c)$ simply contains the newly mined term.

- Let $d'(c)$ denote a set of terms that contains a) the first synonym in the $c$'s synset, b) either the second synonym in $c$'s synset or the first synonym in $c$'s parent's synset. For new concepts, $d'(c)$ simply contains the newly mined term.

In order to assess a potential instance $isA(c_1, c_2)$ (where $c_1$ and $c_2$ are WordNet concepts) our system proceeds as follows: for each $t_i$ in $d'(c_1)$ and $t_j$ in $d(c_2)$, the system computes the probability that $t_i$ is a subclass term for $t_j$. These probabilities are then combined to yield a final probability associated with $isA(c_1, c_2)$ as follows:

$$p(isA(c_1, c_2)) = \frac{\sum_{i=0}^{|d'(c_1)|} \sum_{j=0}^{|d(c_2)|} p(isA(t_i, t_j))}{|d'(c_1)| * |d(c_2)|)} \tag{3.1}$$

As described in the Results section (subsection 3.6.5), the precision of the baseline assessment method is relatively good but the recall is low. In the following, we describe a method for increasing our system's recall.

### *3.6.4 Generating and Assessing Concept-Level IS-A Links: Collective Assessment*

In order to improve its recall, the system uses dependency information (as described below) in order to adjust the initial probabilities assigned to *potential* IS-A instances: correct instances whose initial probability was low but that have enough "support" from related IS-A instances of which we are sure should see their probability boosted. In the following, we describe the dependency information used by our system together with the dependency-based probability update process.

#### *Concept-level dependencies*

Our system will build a set of graphs $G =< V, E >$ that capture the dependencies among related potential IS-A instances - these dependencies will be expressed in terms of dependencies among concepts. We make use of the following concept-level dependency functions:

- $isA_{WN}(c_i, c_j)$ is a function that checks whether $c_i$ is a descendant of $c_j$ with respect to WordNet. The function returns True if and only if:

  1) $c_i$ is a descendant of $c_j$ in the WordNet IS-A hierarchy or

  2) $c_i$ and $c_j$ are compound noun phrases such that I) $c_i$ and $c_j$ have the same head noun and II) the modifiers in $c_i$ and $c_j$ are names of concepts related by means of a

IS-A relationship with respect to WordNet.

An example of such an instance $isA_{WN}(c_i, c_j)$ is $isA_{WN}(\texttt{limousineService}_0, \texttt{carService}_0)$.

If none of the conditions above is met, the function returns False.

- $sibling(c_i, c_j)$ returns True if and only if if $c_i$ and $c_j$ are siblings in WordNet and False otherwise.

- $similar(c_i, c_j)$ is a function whose value is determined as follows: if $c_i$ and $c_j$ co-occur as part of an enumeration or conjunction pattern in $k$ sentences previously extracted by our system from the Web, we compute the estimate $SimPMI(c_i, c_j)$ as follows:

$$SimPMI(c_i, c_j) = \frac{k}{numPatterns(c_i) * numPatterns(c_j)},$$

  where $numPatterns(c_i)$ and $numPatterns(c_j)$ denote the number of sentences containing instantiated enumeration or conjunction patterns that include $c_i$ or, respectively, $c_j$. If the obtained value is higher than a given treshold $t$, the similarity function returns True, otherwise it returns False. The threshold $t$ is automatically estimated using a small set of seed similar and dissimilar concepts obtained from WordNet (two concepts are considered similar if they are siblings in the IS-A hierarchy).

The concept-level dependencies described above are used to determine dependencies among potential IS-A instances that are represented by dependency graphs as seen below.

*Building Dependency Graphs*

Given each target concept, let $c$ represent its ancestor of maximum height that is also a target concept (if no such ancestor exists, $c$ is simply the original target concept.) Let $I$ denote the set of mined concept-level IS-A instances corresponding to $c$ and its descendants.

For each target concept $c$ and instance set $I$, the system builds a *dependency-graph* $G =< V, E >$ as follows:

a) $\forall i \in I$ s.t. $p(i) > 0.5$, add $v_i$ to $V$.

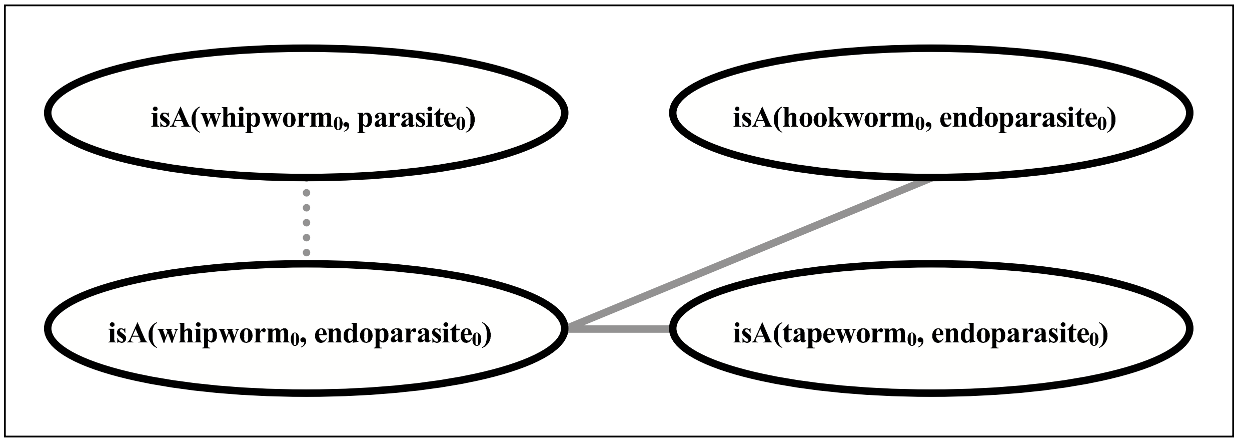

isA(whipworm0, parasite0)
isA(hookworm0, endoparasite0)
isA(whipworm0, endoparasite0)
isA(tapeworm0, endoparasite0)

Figure 3.2: **Example of Partial Neighborhood for Instance** $isA(whipworm_0, endoparasite_0)$**.** Solid lines denote neighbors containing similar concepts (*e.g.*, $whipworm_0$ and $tapeworm_0$) while the dotted line denotes a neighbor related to the vertex of interest by means of a background IS-A relationship ($isA_{WN}(endoparasite_0, parasite_0)$).

b) $\forall i, i' \in I$, $i = isA(c_1, c_2)$, $i' = isA(c_3, c_4)$ s.t. $p(i) > 0.5$, $p(i') > 0.5$, add an undirected edge between $v_i$ and $v_{i'}$ to $E$ ($G$'s edge set) if and only if $i$ and $i'$ satisfy one of the following condition sets: 1) $c_3 = c_1$, $isA_{WN}(c_4, c_2) = True$, 2) $c_3 = c_1$, $isA_{WN}(c_2, c_4) = True$, 3) $c_3 = c_1$, $similar(c_4, c_2) = True$, 4) $c_3 = c_1$, $sibling(c_4, c_2) = True$.

These condition sets define the immediate *neighborhood* of a given vertex (see Figure 3.2 for an example). Each vertex can have one of two labels: *True* or *False*. Intuitively, the label of each node is related to the label of nodes in its immediate neighborhood - in the following, we introduce a set of *neighborhood features* that collectively influence the given node's label.

*Neighborhood Features*

Given the vertex $v_i$ corresponding to the IS-A instance $i = isA(c_1, c_2)$, let $N_i$ denote its corresponding neighborhood. Dependencies between the label of a node and the labels associated with nodes in its neighborhood are used to generate the set of *neighborhood features* that influence the value of a label's node ( Table 3.8 contains the features found to be the most useful). Since we are highly confident in some of the prior probabilities (for

Table 3.8: **Examples of Neighborhood Features For Vertex $v_i$, where $i = isA(c_1, c_2)$.** Notation: $v_i$,$v_j$ = graph vertices, $N_i = v_i$'s neighborhood, $c_1$, $c_2$,$c_m$ = concepts, $p_0(j)$ = the initial value of the probability that $j$ is a correct IS−A instance, $T$ = True, $t, t', t''$ = thresholds.

| **Feature** | **Condition** |
|---|---|
| $f_0(l(v_i) = T) = 1$ iff | $\exists\ v_j \in N_i$ s.t. $j = isA(c_m, c_2)$, $p_0(j) > 0.9$, $isA_{WN}(c_1, c_m)$. |
| $f_1(l(v_i) = T) = 1$ iff | $\exists\ v_j \in N_i$ s.t. $j = isA(c_2, c_m)$, $p_0(j) > 0.9$, $isA_{WN}(c_1, c_m)$. |
| $f_2(l(v_i) = T) = 1$ iff | $\exists\ M \geq t$ nodes $v_j \in N_i$ s.t. $j = isA(c_m, c_2)$, $p_0(j) > 0.9$, $isA_{WN}(c_m, c_1)$. |
| $f_3(l(v_i) = T) = 1$ iff | $\exists\ M' \geq t'$ nodes $v_j \in N_i$ s.t. $j = isA(c_m, c_2)$, $p_0(j) > 0.9$, $sibling(c_m, c_1)$. |
| $f_4(l(v_i) = T) = 1$ iff | $\exists\ M'' \geq t''$ nodes $v_j \in N_i$ s.t. $j = isA(c_m, c_2)$, $p_0(j) > 0.9$, $similar(c_m, c_1)$. |

example, mined IS-A instances with probabilities greater than 0.9 are likely to be correct), we use the corresponding instances in order to compute the feature functions.

The thresholds in Table 3.8 can be either absolute or represent percentages of the neighborhood's size. In our experiments, the neighborhoods were relatively small and so experimentally set absolute thresholds worked fine.

*Update Equation for Node Label Probability*

Given a set of prior probabilities associated with each possible node label, the system iteratively updates these probabilities based on a) the value of this probability computed in the previous iteration and b) the labels of the other nodes in its neighborhood. As mentioned above, we are highly confident in some of the prior probabilities (those over 0.9) and so we focus on updating the probabilities of the mined instances whose prior probabilities are lower than 0.9.

Given node $v_i$ with neighborhood $N_i$ and current label $L$, the probability that $v_i$ has label $L$ ($P(l(v_i) = L)$) is updated from iteration to iteration as follows:

$$P(l(v_i) = L)_{m+1} = \frac{P(l(v_i) = L)_m * (sup(v_i, L)_m)}{P(l(v_i) = L)_m * (sup(v_i, L)_m - sup(v_i, L')_m) + sup(v_i, L')_m}, \qquad (3.2)$$

where $sup(v_i, L)$ and $sup(v_i, L')$ denote the neighborhood support for the two possible labels of node $v_i$ and $m$, $m + 1$ denote the 2 consecutive iterations. This equation is a modified version of the update equation used in the *relaxation labeling* framework described in Chapter 5.

The system stops when the set of most likely labels doesn't change from iteration to iteration.

*Neighborhood Support*

The support function computes the probability of each label for node $v_i$ based on the labels of nodes in $v_i$'s neighborhood $N_i$.

Let $A_k = \{(v_j, L_j) | v_j \in N_i\}$ , $0 < k \leq 2^{|N_i|}$ represent one of the potential assignments of labels to the nodes in $N_i$. Let $P(A_k)_{(m)}$ denote the probability of this particular assignment at iteration $m$. The *support* for label $L$ of node $v_i$ at iteration $m$ is :

$$sup(v_i, L)_{(m)} = \sum_{k=1}^{2^{|N_i|}} P(l(v_i) = L | A_k)_{(m)} * P(A_k)_{(m)}$$

We assume that the labels of $v_i$'s neighbors are independent of each other (this assumption is only made in order to allow for a simple combination function - the labels of these neighbors are in fact related and a more complicated function can be used):

$$sup(v_i, L)_{(m)} = \sum_{k=1}^{2^{|N_i|}} P(l(v_i) = L | A_k)_{(m)} * \prod_{j=1}^{|N_i|} P(l(v_j) = L_j)_{(m)}$$

Every $P(l(v_j) = L_j)_{(m)}$ term is the estimate for the probability that $l(v_j) = L_j$ (which was computed at iteration $m$ using the update equation, as seen above).

The $P(l(v_i) = L | A_k)_{(m)}$ term quantifies the influence of a particular label assignment to $v_i$'s neighborhood over $v_i$'s label - in the following we describe how this term is computed:

Given node $v_i$ with neighborhood $N_i$ and neighborhood boolean feature functions $f_1$, ... $f_k$, we have:

$$P(l(v_i) = L | A_k)_{(m)} = \sum_{t=1}^{k} f_t(L) * w_t$$

where $w_0, ...w_k$ are experimentally set weights that sum up to 1 and reflect the relative importance of the various types of neighborhood features.

### *3.6.5 Preliminary Results and Completion Estimates*

This section describes our preliminary WordNet extension results and offers an estimate of how long it would take to complete a more ambitious WordNet extension. The most expensive part of the WordNet extension process is repeatedly issuing queries to the Google search engine API. In order to reduce the number of needed search engine queries, we use the following settings:

a) Given each seed term, we use 2 extraction rule templates to generate extraction rules.

b) When assessing each subclass term we use only 2 discriminators instead of the entire set of discriminators.

Tables 3.9 and 3.10 contain the results of enriching the WordNet IS-A hierarchy with IS-A links between seed concepts and novel concepts as well as with missing links among existent concepts. The seed set contains 200 concepts; on average, the system extracts 5-6 new links per seed concept if the *baseline assessment* method is used and 7-8 new links per seed concept if the *collective assessment* method is used as well. As we can see from the result tables, most of them are links between seed concepts and newly created WordNet concepts, rather than missing links among WordNet concepts.

*Results: IS-A Links between Existent and Novel Concepts*

Table 3.9: **Results: Augmenting the IS-A Hierarchy With Links Between Existent and Novel Concepts.** The Collective Assessment step increases both recall and precision.

| **Method** | **Precision** | **Recall** | **Avg. Yield Per Target Concept** | **Total Yield** |
|---|---|---|---|---|
| *BaselineAssessment* | 78% | 50.6% | 5-6 | 1141 |
| *CollectiveAssessment* | 82% | 69.3% | 7-8 | 1563 |

In order to evaluate the performance of our system we selected 20 target concepts and proceeded as follows:

a) Given each target concept, we collected all mined potential IS-A links corresponding to this concept (at all probability levels). We had a human judge annotate them as *correct* or *incorrect.*

b) The *precision* of the system is measured as the ratio between the number of *correct* IS-A links proposed by the system and the sum of the *correct* and *incorrect* IS-A links.

c) Since we do not have access to the true set of all IS-A links that could be mined from Web text for a set of target concepts, we evaluate the *recall* of the system as the percentage of the correct (mined) IS-A links (regardless of the associated probabilities) that is retained as part of the system's final output. This recall measure is equivalent with the ratio between the number of true positives and the sum of the number of true positives and false negatives, which is commonly used in information retrieval applications.

The collective assessment method increased the number of extracted IS-A links - the overall precision of the extracted links also improves.

*Results: Missing IS-A Links Among Existent WordNet Concepts*

Table 3.10: **Augmenting the IS-A Hierarchy With Missing Links Among Existent Concepts**. The Collective Assessment step increases both precision and recall.

| **Method** | **Precision** | **Recall** | **Yield** |
|---|---|---|---|
| $BaselineAssessment$ | 74% | 44.4% | 32 |
| $CollectiveAssessment$ | 77% | 70.8% | 51 |

The gold standard for this set of results is obtained in a manner similar to that described above; the precision and recall measures are similarly computed. As above, the collective assessment method results in increased yield and improved overall precision - however, the number of this type of IS-A links remains small.

*Results: Discussion*

Although the precision and recall results are promising, there is room for improvement. The main factor contributing to the error rate is the ambiguity of the used discriminators coupled with the fact that only two discriminators are used. Additional discriminators would reduce the overall ambiguity but they would increase the query requirement. The recall suffers because of limited local context information that can be used to confidently propose a IS-A link: some target WordNet concepts have only one name, a short definition or a parent concept corresponding to an abstraction rarely mentioned in text - since we focus on expanding concepts with few or no descendants, the sparsity of such context information cannot be compensated by taking into account the concept's hyponyms (descendants).

A significant obstacle was represented by the number of discovered compound terms that contained subjective adjectival modifiers and had to be discarded ("fast animals", "angry animals", "healthy animals") - this meant that a lot of the discovery time and extraction queries were spent on finding unusable terms, which greatly impacted the number of usable terms available to the assessment step. This observation was one of the many related experimental results that led other members of our group to recently introduce an in-house search engine [15], so that similar research efforts wouldn't have to contend with using time-consuming commercial search engine queries. The resulting in-house search engine will allow the experiments proposed in this chapter to be executed in a much shorter period of time - one of the members of our group is investigating a similar set of experiments using this newly available resource.

### *3.6.6 Estimates: Completing a Large-Scale WordNet Extension*

Starting with a set of 200 seed WordNet concepts and using the settings described above, we found that the query use per target concept was as follows (see Table 3.11 for a summary):

Table 3.11: **Average Number of Search Engine Queries Per System Component and Target Concept.**

| **WordNet Extension System Component** | **Avg. Query Requirement per Target Concept** |
|---|---|
| Subclass Term Extraction | 22 |
| Subclass Term Assessment | 80 |
| Baseline Assessment of Concept-Level IS-A Instances | 60 |
| Collective Assessment of Concept-Level IS-A Instances | 0 |
| **Total** | **162** |

a) Extracting subclass terms required approximately 22 queries per target concept.

b) Assessing subclass terms required approximately 80 queries per target concept.

c) The baseline assessment of concept-level IS-A instances required approximately 60 additional queries per target concept. In this particular implementation, the collective assessment module did not use any additional queries - more computation intensive versions of this module can use additional Web queries in order to discover additional dependencies among extractions.

In conclusion, the system needed approximately 162 queries per concept and, as previously mentioned, it was able to extract 7-8 new concepts (on average) for each seed concept.

The described WordNet extension can be parallelized since most of the concepts in the seed set can be independently expanded and the proposed expansions for a particular concept can be assessed independently from those for other concepts. The exceptions are represented by concepts whose small number of descendants have also been included in the seed set - as described in subsection 3.6.4, each such concept, together with its descendants, is expanded and the corresponding potential IS-A links are evaluated independently of other concepts in the seed set.

WordNet contains 80,000 concepts named by a noun phrase; in order to *double* the size of this concept set, we would need a seed set of approximately 10,000 concepts and 1620000 queries. We had access to a Google key that allows the use of 100,000 queries per day - expanding each target concept takes, on average, approximately 10 minutes and we found that using 4 machines resulted in no noticeable delays when querying Google (compared with the use of 1 machine). In conclusion, we could expand around 576 concepts a day, which leads to a completion estimate of 17 days (this estimate assumes that the acquisition rate of 7-8 subclass concepts per seed concept holds).

### *3.6.7 Related Work*

Work completed prior to ours (*e.g.*, [3], [41]) had already started addressing the problem of coverage gaps in WordNet. For example, [3] addressed the problem of augmenting the set of core WordNet relations by introducing *topic signatures* that annotate concepts with the name of a topic of interest. [41] investigated the possibility of augmenting WordNet's vocabulary by extracting novel compound terms (*e.g.*, "car service") from domain-specific texts. This latter paper was the most relevant to our work; the proposed extensions were limited and due to the small size of the domain-specific text corpus, the set of features used to evaluate candidate IS-A links was quite large while the basic evaluation mechanism was quite complicated. Since our system leverages the large quantity of available Web text, it is able to obtain good results with a much smaller feature set and much simpler basic assessment mechanism.

Additional related work completed in the same time as ours as well as subsequent investigations of large-scale WordNet extensions validate our approach. Inspired by KnowItAll, a recent Google paper [83] takes advantage of the Google infrastructure in order to augment WordNet with a large number of concept instances as well as with alternative concept definitions (*WordNet glosses*). Another recent paper [85] that follows in KnowItAll's footsteps discusses two methods for ontologizing semantic relations mined at the term level from Web text. While there are similarities between our work and that described in [85] (*e.g.*, both attempt to extend WordNet using relation instances extracted from the Web, both make use of relation-specific lexico-syntactic patterns), there are also many differences: *e.g.*, [85] is concerned with different conceptual relations (Part-Of and Cause) and it uses different feature sets and methods for proposing concept-level instances of target relations.

# Chapter 4
# LEARNING RELATION PROPERTIES

## *4.1 Introduction*

This thesis explores the automatic extraction of detailed domain models from Web text: in addition to extracting subclass and related class information, we are also interested in the automatic acquisition of *relation meta-properties* (*e.g.*, *transitivity*) and *dependencies among relations* (*e.g.*, *entailment*). In addition to being integral to a comprehensive model of a given domain, this information is useful in many applications such as question answering or information extraction systems. In the past, ontology engineers have manually provided this information for each ontology relation of interest; given the recent interest in automatically building large ontologies from textual data, the need for automated methods becomes clear. In this chapter, we describe how relations can be labeled with meta-property and dependency labels using *instance information* in conjunction with informative *lexico-syntactic patterns.*

### *4.1.1 Contributions*

The contributions of this chapter are as follows:

1. We show that the WIE architecture can be instantiated and adapted to the problem of automatically learning relation properties from Web text (see subsection 4.2.1).

2. We report on encouraging preliminary results showing that relation meta-properties and interesting dependencies among relations can be acquired from Web text data (see Section 4.5).

3. We show that relation meta-properties and especially simple dependencies among relations can be used to extract *implicit* factual information from text corpora.

The rest of the chapter is organized as follows: Section 4.2 introduces the main task at hand and gives an overview of our solution, Section 4.3 and Section 4.4 show how relation properties as well as inter-relation dependencies can be learned using a mixture of relation instance data and informative lexico-syntactic patterns, Section 4.5 describes the corresponding experimental results, Section 4.5.3 shows how the automatically acquired information can be used in an information extraction task, Section 4.6 describes related work items and finally, Section 4.7 describes on-going work and outlines some future work items.

**CheckBinaryRelationProperty(R, P, PatternTemplates, f,SE)**

```
1   $p(has_f(R,P)) \leftarrow verifyPropertyUsingFormula(R, f, SE)$;
2   $p(has_p(R,P)) \leftarrow verifyPropertyUsingPatterns(R, PatternTemplates, SE)$;
3   $p(has(R,P)) \leftarrow combine(p(has_f(R,P)), p(has_p(R,P)))$;
```

Figure 4.1: **Overview of Property Learner.** Notation: $R$ = binary relation, $P$ = property of interest, $f$ = formula corresponding to the property of interest, $PatternTemplates$ = set of informative lexico-syntactic pattern templates, $SE$ = search engine

## *4.2 Overview*

This section describes the main task addressed in this chapter and outlines our solution.

*Task*

Given a binary relation $R$ and a property $P$, decide whether $R$ has property $P$.

*Solution*

In the following, we give a brief description of our solution to this problem (see Figure 4.1). Our system assumes access to a search engine $SE$ that can a) retrieve a set of instances for any given relation $R$ and b) retrieve hitcounts for strings of interest. This assumption is reasonable because KnowItAll and WIE can be used to retrieve instances of binary relations (see remarks in Chapter 2) and existent commercial search engines used by WIE compute hit counts for strings of interest. Additionally, the TextRunner relational search engine [15] developed over the past year by members of our research group has the dual functionality mentioned above as well.

We compiled a comprehensive list of binary relation properties (see Table 4.1) and noticed that they can be characterized by simple first order logic formulas involving only the predicates $R(x, y)$ and $Equal(x, y)$ (see subsection 4.2.2 for more details).

Given a binary relation $R$, a property of interest $P$ with a corresponding FOL formula $f$ and a search engine $SE$, the system first checks whether $R$ has property $P$ by evaluating $f$ using the available set of relation instances.

The system then uses a complementary source of information in the form of generic lexico-syntactic patterns in order to estimate the probability that $R$ has property $P$.

A final probability estimate is derived on the basis of these two probability values.

In the following, we discuss the relationship between the property learner and the basic WIE architecture and describe the property learner components in more detail.

### *4.2.1 Using* WIE *Ideas to Learn Relation Properties*

As we can see from the above method description, the property learner adapts the WIE architecture for the problem of automatically learning relation properties.

The property learner uses a search engine to extract relation instances and to compute corpus statistics that reflect the use of property-specific lexico-syntactic patterns instantiated with the information pertaining to the relation of interest. Two assessment steps (one using relational information and the other using lexico-syntactic features) are employed in order to determine the probability that relation $R$ has property $P$ of interest.

Current work items include testing a collective assessment step which takes into account the dependencies among the various properties as well as the dependencies among multiple relations in order to improve the recall of our method.

### *4.2.2 Binary Relation Meta-Properties*

As we can see in Table 4.1, interesting and useful properties of binary relations are described by constraints which can be expressed using FOL (first order logic) formulas - in our case, Horn formulas.

These formulas only use $R(x, y)$ and $Equal(x, y)$; in order to check whether $R(x, y)$ holds, we use the set of instances provided by the available search engine. $Equal(x, y)$ is true if the strings $x$ and $y$ denote the same concept and false otherwise. In our case, given $x$ and $y$, $Equal(x, y)$ has an associated probability denoting how likely $x$ and $y$ are to name

Table 4.1: **Examples of Binary Relationship Properties together with Corresponding Formulas**. Notation: $R \subseteq X \times X$, $R' \subseteq X \times Y$ = binary relations, $Equal(x, y)$ = equality relation. Note: For brevity, we use the notation $\forall x \in X$ ; a well-formed formula would include an additional $elementOfX()$predicate: $\forall x, elementOfX(x) \Rightarrow R(x, x)$.

| **Relation** | **HasProperty** | **Formula** |
|---|---|---|
| $R \subseteq X \times X$ | Is-Reflexive | $\forall x \in X, R(x, x)$ (see Note in caption) |
| $R \subseteq X \times X$ | Is-Transitive | $\forall x, y, z \in X, R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$ |
| $R \subseteq X \times X$ | Is-Symmetric | $\forall x, y \in X, R(x, y) \Rightarrow R(y, x)$ |
| $R \subseteq X \times X$ | Is-Anti-symmetric | $\forall x, y \in X, R(x, y) \wedge R(y, x) \Rightarrow Equal(x, y)$ |
| $R \subseteq X \times X$ | Is-Asymmetric | $\forall x, y \in X, R(x, y) \Rightarrow \neg R(y, x)$ |
| $R' \subseteq X \times Y$ | Is-Injective | $\forall x, y, z \in X, R'(x, z) \wedge R'(y, z) \Rightarrow Equal(x, y)$ |
| $R' \subseteq X \times Y$ | Is-Surjective | $\forall y \in Y, \exists x \in X, R'(x, y)$ |
| $R' \subseteq X \times Y$ | Is-Left-total | $\forall x \in X, \exists y \in Y, R'(x, y)$ |
| $R' \subseteq X \times Y$ | Is-Total | $\forall x, y \in X, R'(x, y) \vee R'(y, x)$ |
| $R' \subseteq X \times Y$ | Is-Bijective | $(\forall x \in X, \exists y \in Y, R'(x, y)) \wedge$ <br> $(\forall y_0 \in Y, \exists x_0 \in X, R'(x_0, y_0)) \wedge$ <br> $(\forall x_1, y_1, z \in X, R'(x_1, z) \wedge R'(y_1, z) \Rightarrow Equal(x_1, y_1))$ |
| $R \subseteq X \times X$ | Is-EquivalenceRelation | $(\forall x \in X, R(x, x)) \wedge$ <br> $(\forall x_1, y_1, z_1 \in X, R(x_1, y_1) \wedge R(y_1, z_1) \Rightarrow R(x_1, z_1)) \wedge$ <br> $(\forall x_2, y_2 \in X, R(x_2, y_2) \Rightarrow R(y_2, x_2))$ |
| $R \subseteq X \times X$ | Is-PartialOrder | $(\forall x \in X, R(x, x)) \wedge$ <br> $(\forall x_1, y_1, z_1 \in X, R(x_1, y_1) \wedge R(y_1, z_1) \Rightarrow R(x_1, z_1)) \wedge$ <br> $(\forall x_2, y_2 \in X, R(x_2, y_2) \wedge R(y_2, x_2) \Rightarrow Equal(x_2, y_2))$ |
| $R \subseteq X \times X$ | Is-TotalOrder | $(\forall x \in X, R(x, x)) \wedge$ <br> $(\forall x_1, y_1, z_1 \in X, R(x_1, y_1) \wedge R(y_1, z_1) \Rightarrow R(x_1, z_1)) \wedge$ <br> $(\forall x_2, y_2 \in X, R(x_2, y_2) \wedge R(y_2, x_2) \Rightarrow Equal(x_2, y_2))$ |
| $R' \subseteq X \times Y$ | Is-Functional | $\forall x \in X, y, z \in Y, R'(x, y) \wedge R'(x, z) \Rightarrow Equal(y, z)$ |
| $R' \subseteq X \times Y$ | Is-Function | $(\forall x \in X, y, z \in Y, R'(x, y) \wedge R'(x, z) \Rightarrow Equal(y, z)) \wedge$ <br> $(\forall x_1 \in X, \exists y_1 \in Y, R(x_1, y_1))$ |
| $R' \subseteq X \times Y$ | Is-1-to-1 | $\forall x, x_1, x_2 \in X, y, y_1 \in Y,$ <br> $(R'(x, y) \wedge R'(x, z) \Rightarrow Equal(y, z)) \wedge$ <br> $(R'(x_1, y_1) \wedge R'(x_2, y_1) \Rightarrow Equal(x_1, x_2))$ |
| $R' \subseteq X \times Y$ | Is-1-to-many | $\forall x \in X, \exists y, y1 \in Y, \neg Equal(y, y1) \wedge R(x, y1) \wedge R(x, y2)$ |

the same concept [1].

Our system checks whether a given relation $R$ has property $P$ by automatically checking the truth of the formula associated with $P$, as seen below.

Table 4.2: **Examples of Relations, Corresponding Meta-properties and Dependencies**.

| **Relation Property or Dependency** | **Examples** |
|---|---|
| Transitive | $locatedSouthOf \subseteq Region \times Region$, $contain \subseteq Substance \times Substance$ |
| Functional | $senatorOf \subseteq (Person \times Year) \times State$<br>$representativeOf \subseteq (Person \times Year) \times State$ |
| 1-to-1 | $presidentOf \subseteq (Person \times Year) \times Country$ |
| Symmetry | $siblingOf \subseteq Person \times Person$, $adversaryOf \subseteq Company \times Company$ |
| Transitive-through | $[locatedIn \subseteq City \times Region, partOf \subseteq Region \times Region]$ |
| Entailment | $[isFullOf \subseteq Substance \times Substance, contain \subseteq Substance \times Substance]$ |
| Equivalence | $[adversaryOf \subseteq Company \times Company, enemyOf \subseteq Company \times Company]$ |

## ***4.3 Instance-based Assessment of Relation Properties***

Given a binary relationship $R \subseteq X \times Y$ and a relational property $P$, we can intuitively check whether $R$ has property $P$ by evaluating the formula $f$ associated with $P$ using the set of relation instances obtained from the available search engine $SE$.

### *4.3.1 Generating and Evaluating Formula Groundings*

Given relation $R$ and formula $f$ corresponding to property $P$ (where $f$ has been converted to CNF), the system first generates a sample set $G$ of formula groundings for $f$ as follows:

a) Let $I$ represent the set of $R$ instances obtained from the search engine $SE$ (we used the TextRunner search engine that returns a set of relation instances together with associated probabilities).

b) Let $G$ represent the set of possible formula groundings obtained by considering all the possible combinations of variable values for the set of variables in the formula.

Given a relation $R$, a formula $f$ in CNF and a particular grounding $g$ of $f$, we compute the probability $p(f(g))$ corresponding to $f$ being satisfied by $g$ as follows:

[1]We assume that an entity resolution algorithm has been applied.

- Each disjunction in $f$ is satisfied if any of its literals is satisfied - we choose the *Noisy-Or* combination function to model this behavior: the probability of a disjunction in $f$ is set to $1 - \Pi_{i=0}^{i=k}(1 - p(l_i))$, where $p(l_i)$ is the probability associated with literal $l_i$.

- The probability that the given grounding satisfies the formula is $p(f(g)) = \Pi_{i=0}^{i=m} p_i$, where $p_i$ is the probability associated with the $i$-th disjunction in $f$.

A grounding $g$ whose associated probability is $p(f(g)) > 0.5$ is referred to as a *true grounding* - all other groundings are referred to as *false groundings*.

**Example** Given binary relation $R$, the transitivity property with corresponding formula $f_{trans}$ (see Table 4.1) and a set of values $g = \{x,y,z\}$ s.t. $p(R(x,y)) = 0.82$, $p(R(y,z)) = 0.90$, and $p(R(x,z)) = 0.85$, we have $p(f_{trans}(\{x,y,z\})) = p((\neg R(x,y) \vee \neg R(y,z) \vee R(x,z))) = 1 - 0.82 * 0.90 * 0.15 = 0.8893$.

As it is to be expected, the method described will not work if the set of known instances for each relation is small (we address this issue later in the chapter).

### *4.3.2 Verifying Property-specific Formulas Using Instance Information*

Let $G^+$ and $G^-$ denote the sets of *true* and, respectively, *false* groundings of $f$.

Given a relation $R$ and a property $P$ with corresponding formula $f$ (in CNF), we compute the following score:

$$score(has_f(R,P)) = \frac{\sum\limits_{g \in G^+} p(f(g))}{\sum\limits_{g \in G^+} p(f(g)) + \sum\limits_{g' \in G^-} p(f(g'))}. \tag{4.1}$$

This score corresponds to the relative support offered by the instance information to the hypothesis that $R$ has property $P$. If this score is greater than a threshold $t$, the system decides that the relation has the property of interest with probability $p$ - the threshold and $p$ are estimated on small development sets.

**Trivial formula satisfiability**

In our case, the binary relations have domain and range sets that are relatively large (the set of all people, the set of all companies, etc.) and therefore the relations are relatively sparse. A random pair of elements $(x, y)$ where $x \in X$ and $y \in Y$ is more likely than not to correspond to a negative instance of $R$ - in other words, $R(x, y)$ will not hold. We inspect the formulas in Table 4.1 and notice that the left-hand side of the implications is likely to be false, which means that the formula will be trivially true.

Given a binary relation $R$ together with a set of complete instances and a property formula $f$, $f$'s trivially true groundings are found in addition to the non-trivially true groundings and the false groundings. When only a sample of $R$'s instances is available, the trivially true groundings may cause the system to mistakenly infer that $R$ has property $P$. In order to avoid this problem, we discard the groundings which render the left-hand side of the implication(s) in a given formula false and compute the support score as follows:

Let $G_n^+$ denote the set of *non-trivial true groundings* of $f$. We have:

$$score(has_f(R, P)) = \frac{\sum\limits_{g \in G_n^+} p(f(g))}{\sum\limits_{g \in G_n^+} p(f(g)) + \sum\limits_{g' \in G^-} p(f(g'))}. \tag{4.2}$$

## 4.4 Assessing Relation Properties With Instance and Lexical Pattern Data

Table 4.3: **Examples of Patterns Correlated With the Presence or Absence of the Functional Property.** Notation : $R \subseteq X \times Y$ = binary relation, $y$, $y_1$, $y_2$ = elements of $Y$, *noPrep*()=function that eliminates a preposition at the end of a relation name, *sg*(),*pl*() = singular/plural forms, $enum(y, y')$ = enumerations containing elements of $Y$.

| **Relation** | **Functional** | **Pattern** | **Example** |
|---|---|---|---|
| $capitalCityOf \subseteq City \times Country$ | yes | $R$ **(*)** sg($Y$) | *capital city of* the *country* |
| $capitalCityOf \subseteq City \times Country$ | yes | sg($Y$)**'s** *noPrep*($R$) | the *country***'s** *capital city* |
| $capitalCityOf \subseteq City \times Country$ | yes | $y$**'s** *noPrep*($R$) | *France***'s** *capital city* |
| $motherOf \subseteq Person \times Person$ | no | *pl*($Y$)**'s** *noPrep*($R$) | *people***'s** *mother* |
| $motherOf \subseteq Person \times Person$ | no | $R$ **(*)** *pl*($Y$) | *mother of* these *people* |
| $motherOf \subseteq Person \times Person$ | no | $R$ **(*)** $enum(y, y')$ | *mother of Joe* and *Laura* |

In addition to instance information a system can use *lexico-syntactic patterns* that are indicative of the presence or absence of a particular property. For example, consider the functional relation $capitalCityOf(City, Country)$: we notice that it occurs in contexts such as "the capital city of this country", "the country's capital city" but very rarely in contexts such as "capital city of these countries".

### *4.4.1 Pattern Templates*

Let $Templates$ denote a set of pattern templates that generate patterns we would expect to see instantiated in text if a relation had a particular property (or lacked that property). An example of a $Functional$ pattern template for a binary relation $R$ with argument types $X$ and $Y$ is: $T = sg(Y)|instanceOf(Y)$**'s** $R$, where $sg()$ is the function computing the singular form of a given noun phrase and $instanceOf(Y)$ indicates an element of $Y$. Table 4.3 contains examples of patterns generated using this template (among others).

Table 4.4: **Examples of Patterns Correlated With the Presence or Absence of the Functional Property.** Notation : $R \subseteq X \times Y$ = binary relation, $y$, $y_1$, $y_2$ = elements of $Y$, *noPrep*()=function that eliminates a preposition at the end of a relation name, *sg*(),*pl*() = singular/plural forms, $enum(y, y')$ = enumerations containing elements of $Y$.

| $sg()$ | $pl()$ | $poss()$ |
|---|---|---|
| $uniqMod$: `the, only, single`, etc. | $nonUniqMod$: `a, some, another, other first, last, second, third` etc. | *noPrep()* |
| $enum(NP, NP')$ | *Adverb* | *Connective* |
| . | , | ; |

*Template Generation* The templates were constructed as follows: starting with a pre-defined set of primitives (such as the singular/plural functions listed in Table 4.4) and constraints on template structure (*e.g.*, must contain exactly 1 mention of the relation name, must contain exactly 2 mentions of the relation name), a set of potential templates were generated and instantiations of these templates were sought in text.

In previous work, people have shown that words or phrases can be cues for semantic relations (for example, "$R$, therefore $R'$ " contains the word "therefore" which is a useful indicator of *verb entailment*). We use a similar idea when compiling the list of the template primitives mentioned above for relation meta-properties:

a) For the *Functional* and *1-to-1* meta-properties, we take advantage of existent work on *noun countability* [6] to compile a set of *uniqueness* modifiers and, respectively, *multiplicity* modifiers. For example, `another` is a uniqueness modifier, while `another` is a multiplicity modifier.

b) For the *Symmetry* and *Transitivity* meta-properties, we focus on identifying *adverbs* or *connectives* that may be correlated with the meta-property of interest (given the previous research, these are the types of phrases that are most likely to be useful). We started with a list of connectives compiled from the linguistic literature and a set of *non-subjective* adverbs (e.g. "wonderfully" is a strongly subjective adverb, whereas "often" is not). The non-subjective adverbs were obtained by starting with a large list of English adverbs and automatically discarding those morphologically related to subjective adjectives (we used a list of strongly subjective adjectives available as a byproduct of Chapter 5, but similar information can be easily obtained from the GeneralInquirer lexical resource [108]).

We used a small set of positive and negative examples of each meta-property and looked on the Web for sentences containing contexts of the form $R$ (*) *connective* $R$, $R$ (*) *adverb* $R$, $R$ (*) $R$ *adverb*, where $R$ denotes a relation or without the meta-property of interest. The connectives and adverbs that were common across relations with the meta-property of interest (and respectively, without this meta-property) were retained. These identified phrases were added to the list of template primitives, as mentioned above.

For each meta-property of interest, the templates found to be instantiated in text for a small set of positive or negative meta-property examples were retained.

Appendix B.1 contains a summary of these potentially useful pattern templates.

### *4.4.2 Checking Meta-properties using Pattern Templates*

Each template $T$ generates a set of lexico-syntactic patterns

$LP = LP_R \bigcup LP_{X,Y,R} \bigcup LP_{X,y,R} \bigcup LP_{x,Y,R} \bigcup LP_{x,y,R}$, where :

a) $LP_R$ corresponds to patterns involving only the name of the relation.

b) $LP_{X,Y,R}$ corresponds to the patterns also involving the names of conceptual classes corresponding to a relation's argument types.

c) $LP_{X,y,R}$ and $LP_{x,Y,R}$ involve a mixture of instance and argument type information.

d) $LP_{x,y,R}$ involves only instance information.

Table 4.3 contains examples of some of these pattern types for the functional property.

Given a relation of interest $R$, each template $T \in Templates$ is associated with a boolean feature $f_T$ whose value is determined based on the quantity $score(T)$ that evaluates the relative support for this template offered by the mentions of the relation and its argument information in text sentences. The score is computed as described below.

Let $Cooc_{X,Y,R} \bigcup Cooc_{X,y,R} \bigcup Cooc_{x,Y,R} \bigcup Cooc_{x,y,R}$ denote the co-occurrences involving the relation name and relation argument information. Let $numCooc$ denote the number of such co-occurrences; let $freq()$ denote the number of times a pattern or a set of patterns are instantiated in the sentences containing such co-occurrences.

We define $score(T)$ as :

$$score(T) = \frac{freq(LP_R) + freq(LP_{X,Y,R}) + freq(LP_{X,y,R}) + freq(LP_{x,Y,R}) + freq(LP_{x,y,R})}{numCooc} \tag{4.3}$$

If $score(T) > t = threshold$, the boolean feature $f_T$ is set to 1, else it is set to 0 (the thresholds can be estimated using a small set of positive and negative examples of the relation property of interest).

The cooccurence information above is obtained from the TextRunner relational engine: for example, a TextRunner tuple or fact $f = [``R'', ``X'', ``Y'']$ yields such information for $R$, $X$ and $Y$.

We also experimented with other ways of converting the counts for various types of patterns into features (by having an individual feature for each pattern type), but data sparsity led us to the cumulative count above.

The features are combined using a Naive-Bayes classifier which computes the final probability value $p(has_p(R, P))$.

### *4.4.3 Checking Meta-Properties using a Mixture of Instance and Pattern Data*

The probabilities outputted by the formula-based and pattern-based classifiers can be used to compute a final probability $p(has(R, P))$ based on the two sources of evidence.

Two boolean features, $F_f$ and $F_p$ are computed by checking whether the respective probabilities are greater than 0.5 or not - if the corresponding probability is greater than 0.5, the feature is set to 1, otherwise is set to 0. The boolean features are combined using a Naive-Bayes computation:

$$p(has(R,P)) = p_0(has(R,P)) * \frac{p(has(R,P)|F_f) * p(has(R,P)|F_p)}{p(F_f) * p(F_p)},$$

where $p_0$ denotes the prior probability associated with the presence of the given meta-property $P$. The meta-properties of interest are relatively rare and we manually initialize the prior probabilities accordingly. The conditional probabilities are estimated using the small sets of positive and negative meta-property examples mentioned in previous subsections.

### *4.4.4 Learning Interesting Dependencies Among Relations*

The same mechanism used to learn relation meta-properties can be employed in order to acquire interesting dependencies among relations *transitive-through*, *entailment*, *equivalence*, etc. Such dependencies can be characterized using the same type of FOL formulas we saw in the previous section (see Table 4.5 for examples). As previously mentioned in the literature, lexical patterns indicative of entailment (especially verb entailment) can be used as well [18] (we show examples of such patterns in Appendix B.1). The appendix also contains examples of patterns corresponding to the Transitive-through relationship. Table 4.5 contains examples of commonly mentioned dependencies involving two relations, but additional dependencies involving three or more relations can be similarly expressed and evaluated.

Table 4.5: **Examples of Dependencies together with Corresponding Formulas**. Notation: $R \subseteq X \times Y$, $R_1 \subseteq X \times Y$, $R' \subseteq X' \times Y'$ = binary relations, $Equal(x,y)$ = equality relation.

| **Relations** | **Dependency** | **Formula** |
|---|---|---|
| $R \subseteq X \times Y$<br>$R' \subseteq X' \times Y'$ | Transitive-through | $\forall x \in X, y \in Y, y' \in Y', R(x,y) \wedge R'(y,y') \Rightarrow R(x,y')$ |
| $R \subseteq X \times Y$<br>$R' \subseteq X' \times Y'$ | Entails | $\forall x \in X, y \in Y, R(x,y) \Rightarrow R'(x,y)$ |
| $R \subseteq X \times Y$<br>$R_1 \subseteq X \times Y$ | Equivalence | $\forall x \in X, y \in Y, (R(x,y) \Rightarrow R_1(x,y)) \wedge (R_1(x,y) \Rightarrow R(x,y))$ |

## *4.5 Results*

This section describes our experimental results and suggests avenues for further investigation. In the following, we refer to the method which uses only instances to assess relation properties as *Baseline* and to the method which uses a mixture of instance data and lexico-syntactic data as *Mixt*. We found that the *Baseline* method works well if relation instances are available and that the use of lexico-syntactic clues by the *Mixt* method significantly improves recall if such instances are scarce or not available.

### *4.5.1 Results: Learning Relation Properties*

In our experiments, we focus on a subset of the meta-properties in Table 4.1: *transitivity*, *symmetry*, the *functional* property and the *1-to-1* property. This subset was chosen based on repeated mentions in the natural language processing and ontology learning literature, but additional meta-properties can be learned in a similar fashion.

*Dataset Construction* The preliminary experiments described in this section used a set of 200 relations chosen from a TextRunner crawl containing a mixture of Nutrition, History of Science and general knowledge information - the relations were annotated with corresponding meta-properties by a human judge (see Table 4.2 for examples of these relations and Appendix B.1 for the comprehensive list).

There are 16 conceptual classes denoting relation argument types; we obtained instances of these conceptual classes using information from the TextRunner hyponym dictionary and WordNet. If the proposed TextRunner hyponyms are proper nouns ("Bill Clinton"), we retain them as instances. If the hyponyms for a particular concept class include very few proper nouns, we use WordNet information to discover *base concepts* in the hyponym list and allow them as well (base concepts C, such as $pear_1$, are concepts whose only hyponyms are proper nouns or compound noun phrases of the type *modifier C*).

Given the 16 classes corresponding to argument types, we populated the 200 relations with corresponding relation instances using TextRunner queries [2]. When these queries yielded only a few or no instances for a particular relation, we used a simplified version of

[2]TextRunner can be accessed at *http://turingc.cs.washington.edu:7125/TextRunner/*

the KnowItAll binary predicate extraction module to identify additional instances on the Web.

As we can see, some of the input relations take as arguments ordered pairs, which in our case are used to model time-dependent relations - the potential properties of such relations are verified in a similar manner with that employed for the simple binary relations.

*Results and Discussion*

We measure the precision and recall of our property learner - Table 4.6 shows the results of our experiments. The *Baseline* method performs well if relation instances are available but is not able to handle relations with few or no instances. The *Mixt* method is able to handle some of these cases due to its use of lexico-syntactic patterns - its recall is on average 7% higher than that of the *Baseline* method. We are in the process of producing an additional set of results on higher-quality data - we will update the exact numbers accordingly, but we expect the overall conclusion to remain valid: combining sources of evidence will enhance the labeler's performance.

Table 4.6: **Results: Automatically Tagging Relations with Corresponding Meta-properties.** Notation: Baseline = instance-based property learning, Mixt = learning properties with a mixture of instance- and lexico-syntactic data. The use of lexico-syntactic data leads to a 7% average increase in recall while the average precision remains almost the same.

| **Property** | **Precision(*Baseline*)** | **Precision(*Mixt*)** | **Recall(*Baseline*)** | **Recall(*Mixt*)** |
|---|---|---|---|---|
| Transitivity | 81% | +2% | 62% | +9% |
| Functional | 73% | -2% | 68% | +6% |
| 1-to-1 | 70% | +2% | 54% | +9% |
| Symmetry | 84% | +1% | 74% | +5% |
| **Average** | **77%** | **+1%** | **64.5%** | **+ 7%** |

## *4.5.2 The Importance of Relation Argument Types*

Throughout this chapter we assume that our system takes as input a set of binary relations with known argument types - the reason is that many relations are not properly defined, and therefore cannot be analyzed, in their absence. For example, *vice-president* may name the relation *vice-president(Person, Country)* or the relation *vice-president(Person, Organization)*. These two relations have different properties: for example, a country typically has only one vice-president whereas an organization may have multiple vice-presidents.

Nevertheless, we noticed as part of our experimental evaluation that the specific argument types of relations that are *symmetric* or *transitive* matter less than those of *functional* or *1-to-1* relations. As part of a separate experiment, the symmetry property and transitive property were reliably identified for relations after omitting the specific type information (and simply declaring that each relation is defined over the *Entity* set). The average recall for these two properties went up by 8%, while precision went up by 4% - upon inspection, this dual improvement is due to the fact that more relation instances were available to the instance-based property learner and that patterns that rely on the occurrence of the relation name in a certain context (omitting argument type information) are useful for the meta-properties at hand. Intuitively, much of the information about the relation's symmetric or transitive properties can be inferred based on the relation name (*e.g.*, *meet with*, *locatedIn*, *relatedTo*).

Omitting specific argument type information when assigning functional and 1-to-1 property labels leads to 18% precision (albeit slightly increased recall). Overall, specific argument type information is quite useful, although we see that symmetric and transitive relations can be discovered even in its absence.

### *Using Relation Properties: Ongoing Work*

Members of our research group are currently working on using relation properties as part of the feature sets for two tasks: a) entity resolution and b) improving the performance of open-domain information extraction (as embodied by the TextRunner system).

Intuitively, when trying to decide whether two relation descriptions refer to the same underlying relation, having knowledge of the relations' *property profiles* can be very useful: if the two relations have different properties, they are less likely to be equivalent.

The second effort is looking at using relation property information (especially the functional and 1-to-1 property information) in conjunction with other types of ontological information in order to improve the performance of information extraction in TextRunner. Intuitively, knowing that a relation is functional or 1-to-1 is helpful for improving precision and knowing that is symmetric or transitive can help improve the number of extracted instances (hence, the recall of the system).

Table 4.7: **Results: Automatically Acquiring Dependencies Among Relations.** Notation: *Baseline* = instance-based property learning, *Mixt* = learning properties with a mixture of instance- and lexico-syntactic data. The use of lexico-syntactic data leads to a 12% average increase in recall and a 4% average increase in precision

| **Dependency** | **Precision** *Baseline* | **Precision** *Mixt* | **Recall** *Baseline* | **Recall** *Mixt* |
|---|---|---|---|---|
| Transitive-through | 86% | +4% | 64% | +14% |
| Entailment | 83% | +3% | 68% | +11% |
| Equivalence | 73% | +5% | 73% | +12% |
| **Average** | **81%** | **+4 %** | **68%** | **+12%** |

.

*Learning Interesting Dependencies*

In addition to labeling relations with meta-properties, we also look at extracting dependencies among relations. In our experiments, we focus on *transitive-through*, *entailment* and *equivalence* (see Table 4.5). The preliminary experiments described in this section used a set of 100 frequent, open-domain relations, obtained from a general TextRunner crawl (see Table 4.2). The relations were annotated with dependencies of interest by a human judge.

Table 4.7 shows the results of our experiments. While the baseline method performs well, the *Mixt* method's recall is on average 12% higher than that of the *Baseline* method while the precision is on average 4% higher.

### *4.5.3 A Task-based Evaluation for Relation Interdependencies*

While relation meta-properties and inter-relation dependencies are interesting from a domain modeling perspective, our interest in this information is due to the following observation: information extraction systems have typically focused on extracting facts *explicitly* spelled out in text but a large number of facts are *implicitly* stated in a given sentence, paragraph or text corpus; such facts can be inferred from a combination of explicitly stated facts and background rules describing the relation properties and the dependencies among the various domain relations.

In this section, we describe how this scenario is used to evaluate the automatically learned relation-specific information.

*Extracting Implicit Facts from Text Corpora*

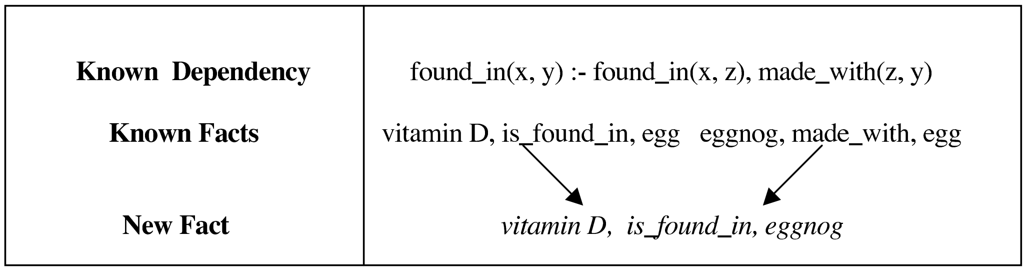

**Known Dependency**
found_in(x, y) :- found_in(x, z), made_with(z, y)
**Known Facts**
vitamin D, is_found_in, egg
eggnog, made_with, egg
**New Fact**
*vitamin D, is_found_in, eggnog*

Figure 4.2: **Example of Implicit Factual Information.**

In the following we propose a *two-pass fact extraction* procedure for a given text corpus.

I. A first pass extracts *explicitly stated* facts from each sentence in the corpus.

A *relation property and inter-relation dependency learning* step uses the methods described in 4.3 and 4.4 to acquire a set of relation meta-properties and inter-relation dependencies - this step focuses on frequently mentioned relations.

II. A second extraction pass examines the corpus anew, using the extracted *explicit* facts and the learned relation-specific information in order to extract *implicit* facts. Facts can be implicitly stated at the sentence, paragraph, article or corpus level - in our preliminary experiments we looked at extracting implicit facts at corpus level, but the experiments can be repeated with a focus on a specific set of articles or article paragraphs.

In these experiments we focused on *concrete facts* (*e.g.*, [*vitamin D, is found in, egg*]) rather than *abstract facts* (*e.g.*, [*vitamins, are found in, foods*])[3].

In the following we describe how implicit facts are extracted from a body of text using a combination of known facts, relation properties (specifically, symmetry and transitivity) and inter-relation dependencies. For example, Figure 4.2 shows how the implicit fact [*vitamin*

[3]This terminology is consistent with that used in the TextRunner project [15].

*D, is found in, eggnog*] can be established based on the explicit facts [*vitamin D, is found in, egg*] and [*egg, is found in, eggnog*].

### *4.5.4 Generating Potential Implicit Facts*

Given a text corpus, potential implicit facts are generated as follows:

For each pair of entities $(e_1, e_2)$ mentioned in the corpus and each relation $R \subseteq X \times Y$, a potential fact $f = (R, e_1, e_2)$ is generated if the following conditions are met:

a) $e_1$ and $e_2$ are instances of $X$ and, respectively, $Y$,

b) $e_1$, $e_2$ and $R$ co-occur in at least 1 sentence or

c) $e_1$ and $R$ and, respectively, $e_2$ and $R$ co-occur in at least 1 sentence.

The above conditions ensure that the generated potential fact is well-formed (condition a)) and try to eliminate unlikely candidates by focusing on entities and relations that co-occur in text rather than on combinations of entities and relations that seem to have no connection (conditions b) and c) ).

### *4.5.5 Assessing Potential Implicit Facts*

Given a collection of explicit facts $F$, a set of rules $T$ describing relation meta-properties and relation interdependencies and a potential fact $f$, the system finds the set of proof trees for $f$ in the context of $F$ and $T$ (in our experiments, we limit ourselves to trees of *height* $\leq 3$). If there are no such proof trees, the fact is considered false. The obtained proof trees are used to instantiate a very simple Bayesian network that estimates the probability of the potential fact $f$ - the predictions of multiple parent nodes are combined using the Noisy-OR combination function (this type of instantiation is common in *knowledge-based model construction* frameworks). If the final probability associated with the True label for the target fact is greater than 0.5, the fact is labeled True, otherwise it is labeled False.

### *4.5.6 Preliminary Results: Extracting Implicit Facts from Text Corpora*

In our experiments, we used a Nutrition text corpus generated by the TextRunner system by crawling the Web; TextRunner also performs an initial extraction of *explicit facts* from

Table 4.8: **Preliminary Results: Extracting Implicit Facts in the Nutrition Domain.** Relation interdependencies prove useful in assessing potential implicit facts with high accuracy.

| **Potential Facts** | **Size** | **Precision** | **Accuracy** | **Recall** |
|---|---|---|---|---|
| Actual Facts | 176 | 74% | 71% | 64% |
| Non-facts | 124 | 88% | 83% | 82% |
| Overall | 300 | 81% | 77% | 73% |

each sentence. As part of this set of initial experiments, we focused on acquiring the meta-properties and inter-relation dependencies for a set $S$ of 100 frequent relations (see Section 4.5). The system automatically generated a set of potential implicit facts (by generating a set of potential facts as described above and eliminating all those already in the set of extracted explicit facts) and restricted it to potential implicit facts involving relations in the set $S$ above.

A human judge manually labeled 300 elements of this set as actual facts or non-facts: there were 176 actual facts and 124 non-facts.

*Results and Discussion* Our system has an average labeling accuracy of 77% - the system recognizes actual facts with 71% accuracy and non-facts with 83% accuracy (see Table 4.8 for a summary of these results). For the purpose of this experiment, we made the *closed world assumption* which resulted in non-facts being recognized with good precision. However, the majority of potential implicit facts will end up being non-facts in most knowledge domains - finding actual implicit factual information is of greater importance.

The factors that led to decreased precision were:

a) the presence of incorrect interdependencies: $fight(X, Z) : -cause(X, Y), lead_to(Y, Z)$. This rule is responsible for conclusions such as *[milk, fights, heart attack]* based on the facts *[milk, causes, heart disease]* and *[heart disease, leads to, heart attack]*. The incorrect interdependencies are in turn extracted based on either erroneous TextRunner-extracted relation instances or, more often, on using *abstract facts* instead of *concrete facts* when learning interdependencies. An example of such an abstract fact is *diet, contains, vitaminC* - using this fact as an instance of the Contain relation is incorrect and will lead to incorrect generalizations or relation interdependencies.

b) the presence of erroneous TextRunner facts and the incorrect substitution of abstract facts for concrete facts when computing the proof trees.

The factors that led to decreased recall are the sparsity of the relation instance data and, as a result, the lack of rules that may have proven useful when computing the final proof trees. With this in mind, our current work focuses on finding additional high-quality actual facts by extracting additional types of relation interdependencies.

## ***4.6 Related Work***

The ontology editing, modeling and learning literature is quite large - in the following we will first focus on the papers most relevant to our work and then briefly mention other related recent efforts.

*Learning Relation Properties* The paper most relevant to ours is [120], which looks at automatically annotating ontology concepts with a specific set of meta-properties (*rigidity*, *stability*, *identity* and *unity*). These meta-properties are useful for a recently introduced ontology evaluation mechanism (OntoClean [43]) which automatically detects ontology modeling errors (such as incorrect subclasses) by checking whether the three meta-properties involved satisfy a specified set of constraints. Inspired by KnowItAll and previous NLP work, [120] investigates the use of lexical patterns in conjunction with Web data in order to eliminate the need for human annotation of concepts with the relevant meta-properties. The initial version of this chapter was independently completed in 2005 in parallel with this related work - in our case, the use of Web statistics and lexical patterns for this task seemed like a natural follow-up to the work on learning class extraction from the Web. While our approaches have similar elements, there are also many differences: 1) we examine a different set of meta-properties, 2) unlike [120], we use a mixture of instance data and lexical pattern data, 3) our lexical pattern feature set is richer than theirs and makes use of instance information, 4) our assessment mechanism is different than that of [120] and finally, 5) we are currently investigating the effect of dependency-based update modules on this task.

Recent efforts to improve the representation of ontology axioms (which include property-specific axioms such as those in Table 4.1) include [127] and [23]. [127] introduces an ontology editor which has been built to support ontology development with help of inferencing and development of ontology axioms. This editor builds on previous work ([107]) that examines ways of representing ontology axioms - while the papers describe a representation and an architecture that allows axioms to be checked after being elicited from users, they do not contain an experimental evaluation. Similarly, [23] focuses on eliciting axioms from ontology users by presenting them with (manually created) easy-to-understand and easy-to-use templates.

*Learning Interesting Dependencies Among Relations*

The most relevant related paper concerning the learning and use of interesting dependencies among relations is [27], which looks at automatically learning simple axioms from a noisy textual database. The axioms are used to clean up the database by eliminating some of the incorrect extractions - additionally, the axioms can be used to extract additional facts from text. While some of the learned axioms are similar to those discussed in this chapter, the paper's approach is limited by its reliance on noisy instance data. Our approach looks at augmenting this source of information with lexico-syntactic patterns; furthermore, [27] treats the simple learned dependencies as merely additional patterns rather than FOL rules that can be used to retrieve proof trees.

In the past two years, other researchers (*e.g.*, [18], [112]) have looked at the potential of lexical patterns to uncover semantic relations (such as entailment) between verbs - our work combines lexical patterns with instance information.

## *4.7 Ongoing Work*

We are currently investigating the effect of collective assessment on the performance of the property learner described in this chapter: we are exploring the use of various types of dependencies in order to improve the performance of the property learner: dependencies among relations (such as entailment or cluster membership for a set of automatically computed clusters) as well as dependencies among the meta-properties of interest (a transitive relation is not 1-to-1).

In addition to investigating the effect of a dependency-based update module on the performance of the property learner, we are also exploring a different collective assessment scheme based on the URNS model introduced in [31]. This model was developed in order to predict more accurate probabilities associated with an extraction being a potential instance of a given concept class - in our case, the model will predict a probability associated with a particular relation being a member of the set of relations with property P. Additionally, we are in the process of testing a set of relation entailment measures devised on top of the URNS model (*under preparation*).

# Chapter 5
# EXTRACTING PRODUCT FEATURES AND OPINIONS FROM REVIEW DATA

This chapter describes our work on applying the ideas behind the information extraction architecture at the core of this thesis to the problem of *in-depth review mining.* In the following, we introduce the problem at hand and give an overview of the chapter's contributions and structure.

## *5.1 Introduction*

The Web contains a wealth of opinions about products, politicians, and more, which are expressed in newsgroup posts, review sites, and elsewhere. As a result, the problem of "opinion mining" has seen increasing attention over the past three years from [116, 49] and many others. This chapter focuses on product reviews, though we plan to extend our methods to a broader range of texts and opinions.

Product reviews on Web sites such as `amazon.com` and elsewhere often associate metadata with each review indicating how positive (or negative) it is using a 5-star scale, and also rank products by how they fare in the reviews at the site. However, the reader's taste may differ from the reviewers'. For example, the reader may feel strongly about the quality of the gym in a hotel, whereas many reviewers may focus on other aspects of the hotel, such as the decor or the location. Thus, the reader is forced to wade through a large number of reviews looking for information about particular features of interest.

This chapter introduces OPINE, an unsupervised information extraction system which mines and summarizes product review information. We decompose the problem of review mining into the following main subtasks:

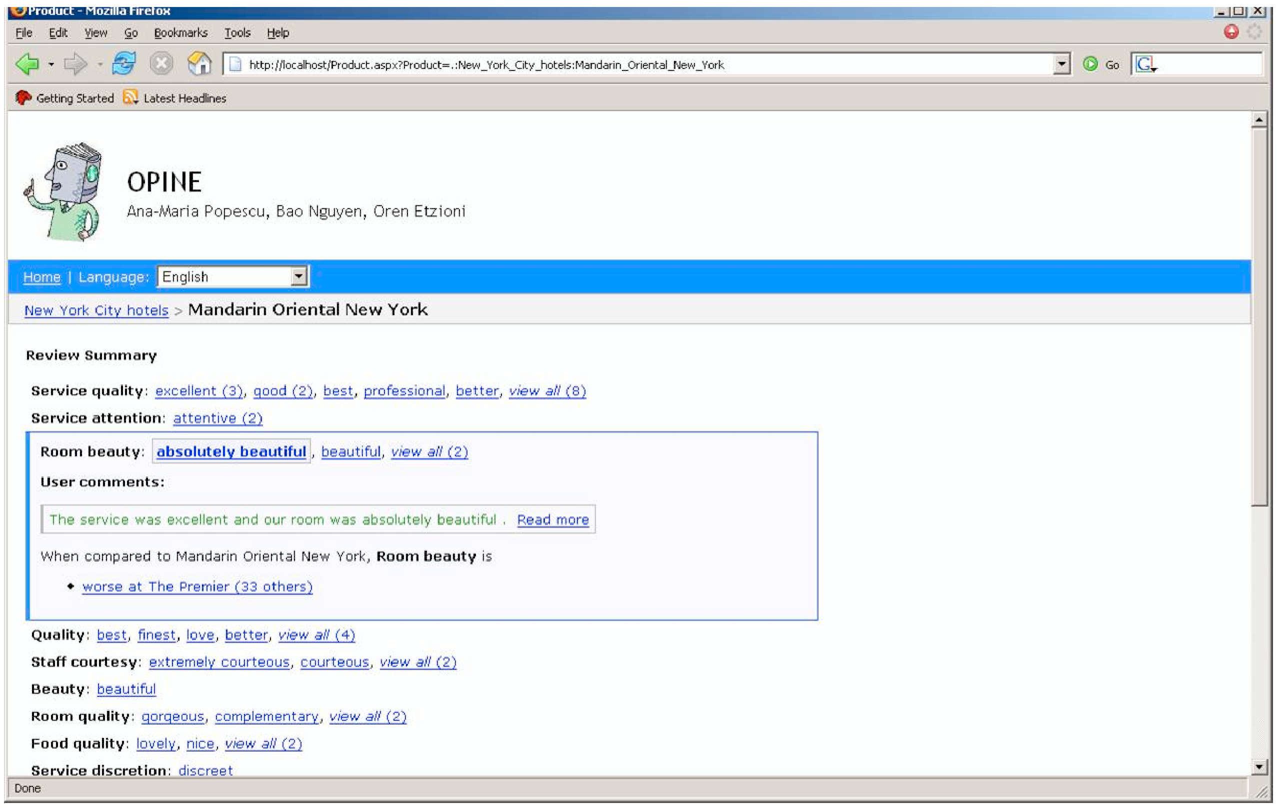
**I. Identify product features**. In a given review, features can be *explicit* (*e.g.*, "the *size* is too big") or *implicit* (*e.g.*, "the scanner is slow" refers to the "scanner speed").

**II. Identify opinions regarding product features**. For example, "the size is *too big"* contains the opinion phrase *"too big"*, which corresponds to the "size" feature.

**III. Determine the polarity of opinions**. Opinions can be *positive* (*e.g.*, "this scanner is *so great"*) or *negative* (*e.g.*, "this scanner is a *complete disappointment"*).

**IV. Rank opinions based on their strength**. For example, *"horrible"* is a stronger indictment than *"bad"*.

OPINE is an unsupervised information extraction system which embodies solutions to all of the above subtasks.


Product - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/Product.aspx?Product=.:New_York_City_hotels:Mandarin_Oriental_New_York
Go
Getting Started Latest Headlines
Done

OPINE

Ana-Maria Popescu, Bao Nguyen, Oren Etzioni

Home | Language: English

New York City hotels > Mandarin Oriental New York

**Review Summary**

**Service quality**: excellent (3), good (2), best, professional, better, *view all* (8)

**Service attention**: attentive (2)

**Room beauty**: **absolutely beautiful**, beautiful, *view all* (2)

**User comments:**

The service was excellent and our room was absolutely beautiful . Read more

When compared to Mandarin Oriental New York, **Room beauty** is

- worse at The Premier (33 others)

**Quality**: best, finest, love, better, *view all* (4)

**Staff courtesy**: extremely courteous, courteous, *view all* (2)

**Beauty**: beautiful

**Room quality**: gorgeous, complementary, *view all* (2)

**Food quality**: lovely, nice, *view all* (2)

**Service discretion**: discreet

Figure 5.1: **Opinion Summary for the Mandarin Oriental New York Hotel**

Figure 5.1 shows the opinion summary generated from a set of customer reviews for the Mandarin Oriental New York hotel. Information about the polarity and relative strength of opinions is used to compare different hotels with respect to a particular feature. For example, the rooms at the Mandarin Oriental New York are more beautiful than those at 33 other hotels.

### *5.1.1 Contributions*

The contributions of this chapter are as follows:

1. We show that the ideas underlying the WIE information architecture can be used to solve various subtasks of review mining.

2. We compare OPINE with the review mining system of Hu and Liu [49] and find that OPINE's precision on the *feature extraction* task is 22% higher than that of Hu&Liu, although its recall is 3% lower. We show that 1/3 of OPINE's increase in precision comes from the use of its *feature assessment* mechanism on review data while the rest is due to Web statistics.

3. We describe OPINE's novel use of a *relaxation labeling* method to find the semantic orientation of words in the context of given product features and sentences.

4. While many other systems have used extracted opinion phrases in order to determine the polarity of sentences or documents, OPINE reports its precision and recall on the tasks of *opinion phrase extraction* and *opinion phrase polarity extraction in the context of known product features and sentences.* On the first task, OPINE has a precision of 79% and a recall of 76%. On the second task, OPINE has a precision of 86% and a recall of 89%.

5. Additionally, OPINE ranks the opinion phrases corresponding to a particular property based on their strength and registers a precision of 73% (according to an evaluation of its output by a human judge).

6. Finally, we report good preliminary results on the problem of extracting *opinion sentences* and establishing their positive or negative character.

The rest of this chapter is organized as follows: Section 5.2 introduces the terminology used throughout this chapter, Section 5.3 describes how our information extraction model is used to handle review mining, Sections 5.4 through 5.9 give an overview of OPINE, describe and evaluate its main components in detail, Section 5.10 discusses related work and Section 5.11 contains our conclusions and future work directions.

## *5.2 Terminology*

A *product class* (*e.g.*, Scanner) is a set of *products* (*e.g.*, Epson1200). OPINE extracts the following types of *product features*: *properties*, *parts*, *features of product parts*, *related concepts*, *parts* and *properties of related concepts* (see Table 5.1 in subsection 5.5 for examples in the Scanner domain). *Related concepts* are concepts relevant to the customers' experience with the main product (*e.g.*, the company that manufactures a scanner). The relationships between the main product and related concepts are typically expressed as verbs (*e.g.*, "the company *manufactures* scanners") or prepositions ("scanners *from* Epson"). Features can be *explicit* ("good `scan quality`") or *implicit* ("good scans" implies good `ScanQuality`).

OPINE also extracts *opinion phrases*, which are adjective, noun, verb or adverb phrases representing customer opinions. Opinions can be *positive* or *negative* and vary in *strength* (*e.g.*, "fantastic" is stronger than "good").

## *5.3 Review Mining with* WIE

OPINE was built by instantiating and adapting the WIE architecture for the case of in-depth product review mining. Potential product features are extracted from review text and then assessed using a classifier whose features are generated based on instantiated lexico-syntactic patterns and corresponding Web-scale statistics. Our experimental results show that the Web-scale computation of the statistics significantly contributes to the high precision of the extracted features, thereby validating this key component of our underlying architecture.

The WIE architecture is further validated by OPINE's ability to identify with high precision the semantic orientation of feature-specific opinion phrases. Potential opinion phrases are first extracted from product review data and their semantic orientation is then computed by a collective assessment step that uses a mix of Web-scale statistics, statistics computed over the corpus of reviews and information gleaned from outside resources (in this case, WordNet). Once again, the ideas underlying the WIE architecture - the use of lexico-syntactic patterns, corpus statistics and the inclusion of a collective assessment step - prove useful in the context of a specific information extraction task.

Due to the specific requirements of our review mining application (outlined in the Introduction section), we introduce an additional module that computes a strength-based ordering of opinions for each feature in the extracted product feature set. While this module is not part of the basic WIE architecture, it benefits from key WIE ideas: instantiated lexico-syntactic patterns are used to determine the relative strength of two opinion phrases corresponding to the same underlying property. These partial strength-based orderings of the opinion phrases are then used to determine a total ordering of the given opinion phrase set for a specific product feature.

In conclusion, the OPINE system represents an instantiation and extension of our core WIE architecture - the applicability of fundamental WIE ideas to the various review mining subtasks strengthens our claim that WIE is a useful and extensible information extraction architecture.

## *5.4* OPINE *Overview*

This section gives an overview of OPINE (see Figure 5.2) - its components are described in more detail in the following sections.

Given product class $C$ with instances $I$ and corresponding reviews $R$, OPINE's goal is to find a set of (feature, opinions) tuples $\{(f, o_i, ...o_j)\}$ such that $f \in F$ and $o_i, ...o_j \in O$, where:

a) $F$ is the set of product class features in $R$.

b) $O$ is the set of opinion phrases in $R$.

c) $f$ is a feature of a particular product instance.

d) $o$ is an opinion about $f$ in a particular sentence.

d) the opinions associated with $f$ are ranked based on opinion strength.

The steps of our solution are outlined in Figure 5.2 above. OPINE parses the reviews using MINIPAR [61] and implements a simple pronoun resolution algorithm which is then run on the parsed review data. The pronoun resolution uses a set of simple rules (such as *number* agreement rules ) compiled from the pronoun resolution literature (*e.g.*, [101]) in order to find pronouns that can be confidently resolved - if a pronoun is ambiguous in the context of a sentence, we do not currently resolve it.

The resulting version of the data is used to find *explicit* product features. OPINE's

**Input**

product class C, reviews R, parser P, search engine S,

generic meronymy lexico-syntactic patterns PT,

generic rules RL for extracting potential opinions,

generic rule templates RT for finding phrases with related semantic orientations,

*WordNet lexical ontology WN and WordNet-based adjective similarity rules RS (optional).*

**Output**

set of [feature, ranked opinion list] tuples.

**ExtractFeaturesAndOpinions**

```
R' ← parseReviews(R,P);
E ← findExplicitFeatures(R', C, S, PT);
O ← findOpinions(R', E, S, RL, RT, WN);
CO ← clusterOpinions(O, RS, S, WN);
I ← findImplicitFeatures(CO, E, R', S, WN);
RO ← rankOpinionsBasedOnStrength(CO);
{($f$, $o_i, ...o_j$)...}←outputTuples(RO, I ∪ E);
```

Figure 5.2: **OPINE Overview.** OPINE takes as input a product class C, a corresponding set of reviews R, a parser P and a small set of patterns or rule templates for each of the review mining subtasks addressed. The system assumes access to a search engine S and to the WordNet lexical ontology (the latter is optional).

*Feature Assessor* and its use of Web PMI statistics are vital for the extraction of high quality features (see 5.5.1). OPINE then identifies *opinion phrases* associated with explicit features and finds their polarity. OPINE's novel use of relaxation labeling techniques for determining the semantic orientation of potential opinion words in the context of given features and sentences leads to high precision and recall on the tasks of *opinion phrase extraction* and *opinion phrase polarity extraction* (see 5.7).

Opinion phrases refer to *properties*, which are sometimes *implicit* (*e.g.*, "tiny phone" refers to the size of the phone). In order to extract implicit properties, OPINE first clusters

Table 5.1: **Explicit Feature Information**

| **Explicit Features** | **Examples** | **% Total** |
|---|---|---|
| Properties | ScannerSize | 7% |
| Parts | ScannerCover | 52% |
| Features of Parts | BatteryLife | 24% |
| Related Concepts | ScannerImage | 9% |
| Related Concepts' Features | ScannerImageSize | 8% |

opinion phrases (*e.g.*, *tiny* and *small* will be placed in the same cluster), automatically labels the clusters with property names (*e.g.*, *Size*) and uses them to extract *implicit* features (*e.g.*, `PhoneSize`). The final component of our system is the ranking of opinions that refer to the same property based on their strength (*e.g.*, $fantastic$ > *great* > *good*). Finally, OPINE outputs a set of (feature, ranked opinions) tuples for each identified feature.

In the following, we describe each of these steps in more depth.

## 5.5 Finding Explicit Features

OPINE extracts *explicit* features for the given product class from parsed review data. The system recursively identifies the *parts* and the *properties* of the given product class and their parts and properties, in turn, continuing until no more such features are found or for a set number of iterations (we found that 1 or 2 iterations were usually enough to find most of the features of this type). The system then finds *related concepts* and extracts their meronyms (parts and properties). Table 5.1 shows that each feature type contributes to the set of final features (averaged over 7 product classes, as described in Section 5.5.1 ).

In order to find parts and properties, OPINE first extracts the noun phrases from reviews and retains those with frequency greater than an experimentally set threshold $t$. OPINE's *Feature Assessor*, which is an instantiation of KnowItAll's Assessor, evaluates each noun phrase by computing the PMI scores between the phrase and *meronymy discriminators* associated with the product class (see Table 5.2 for possible discriminators). The system first uses the review corpus to compute the PMI scores; the explicit features identified in this manner are set aside and the rest of the potential features are assessed using the Web.

If necessary, OPINE is able to distinguish parts from properties using WordNet's IS-A hierarchy (which enumerates different kinds of properties) and morphological cues (*e.g.*, "-iness", "-ity" suffixes). Distinguishing between parts and properties of a product can be useful for ontologies such as the ones currently being built by the CALO project [1] ( OPINE represents the contribution of the University of Washington team to this project).

**Meronymy Discriminators** OPINE primarily uses three high precision meronymy lexical patterns (*[M] of [C]*, *[C] has [M]*, *[C]'s [M]*) that were previously introduced in the NLP literature [9]. The experimental section of this chapter reports on good results obtained using this small set of generic lexical patterns and leveraging the Web corpus - however, one of the active areas of this project is better leveraging the available review corpus by learning additional domain-specific meronymy patterns and using their instantiations in the review corpus to assess a potential meronym.

In order to learn additional meronymy patterns, the system considers a sample of the meronyms acquired using the generic patterns and proceeds as described below. Let $C$ denote the product class of interest: given a parsed sentence $s$ that contains C (or an instance of C) and a known meronym M, the system finds the shortest path from C to M in the parse tree and retains the words along the path in order to form a new lexical pattern.

For example, consider the product class C = `Scanner`, the meronym M = `DocumentFeeder` and the sentence: "The 1640SU-OFFICE `Scanner` is equipped with a `document feeder` that scans up to 30 pages of legal size documents".

Based on this sentence, the system generates the lexical pattern "[C] (be) equipped with [M]". If at least $k$ paths involving different meronyms generate the same pattern $P$, $P$ is added to the set of meronymy lexical patterns (see Table 5.2). We noticed that even if some of these patterns are two general as meronymy patterns, they are still very useful as general *feature* extraction patterns (encompassing meronyms *and* related concepts).

**Related Concepts** Given a target product class $C$, OPINE finds concepts related to $C$ by extracting frequent noun phrases as well as noun phrases linked to $C$ or $C$'s instances through verbs or prepositions (*e.g.*, "The `scanner` *produces* great `images`"). *Related concepts* are assessed as described in [92] and then stored as product features together with their parts and properties.

Table 5.2: **Meronymy Lexical Patterns** Notation: $[C]$ = product class (or instance), $[M]$ = candidate meronym $(*)$ = wildcard character

| | |
|---|---|
| $[M]$ of (*) $[C]$ | $[C]$ has (*) $[M]$ |
| $[C]$'s M | $[M]$ for (*) $[C]$ |
| $[C]$ with (*) $[M]$ | $[C]$ contain(s)(ing) (*) $[M]$ |
| $[C]$ equipped with (*) $[M]$ | $[C]$ endowed with (*) $[M]$ |
| $[C]$ offer(s)(ing) (*) $[M]$ | $[C]$ boast(s)(ing) (*) $[M]$ |

*Extracting Explicit Features: Running Time Discussion*

Let $S$ represent the set of review sentences of interest and let $len$ represent the maximum sentence length. Let $PF$ represent the set of potential features - $|PF|$ is at most $len * |S|$. Finally, let $t_{assess}$ denote the time it takes to check that a noun phrase $f$ is a feature of a given concept class $C$ ($t_{assess}$ is a small constant).

The running time of the explicit feature extraction step is as follows:

a) Parsing the available sentences, running the pronoun resolution module and recording frequency information for identified noun phrases is done in $O(|S| * len^3)$ time (the bulk of this time is spent on computing the full parse for each sentence). In practice, the parser includes a large number of optimizations that allow the parsing to be quite fast. Once frequency information is available, potential features are extracted at no additional cost.

b) Given the product class $C$, finding its direct features takes $O(|S| * len^3 + |PF| * t_{assess}) = O(|S| * len^3 + |S| * len)$ time, but the parser optimizations mean the time is closer to $O(|S| * len^2)$ or even $O(|S| * len * log(len))$.

If finding derived features is necessary, the worst case running time for this additional step will be $O(|PF|^2 * t_{assess})$ - however, in practice we implement a number of optimizations that greatly reduce this time:

1) Direct features that are frequently mentioned yield most derived features, so the system can consider only the most frequent $k$ direct features in step 2) below;

2) Related features tend to be discussed in close proximity: given a direct feature $f$, let $PF'$ represent the size $k'$ subset of potential features that most frequently cooccur with it within a size $m$ window - $k'$ is an experimentally set constant that is much smaller than $|PF|$. Computing $PF'$ can be done in $O(k * log(k))$ - where $k$ is the number of direct

Table 5.3: **Precision Comparison on the Explicit Feature Extraction Task.** OPINE's precision is 22% better than Hu's precision; Web PMI statistics are responsible for 2/3 of the precision increase. All results are reported with respect to Hu's.

| **Dataset** | *Hu* | **Hu Assess(Reviews)** | **Hu Assess(Reviews,Web)** | **OPINE (Reviews)** | **OPINE** |
|---|---|---|---|---|---|
| $D_1$ | *0.75* | +0.05 | +0.17 | +0.07 | **+0.19** |
| $D_2$ | *0.71* | +0.03 | +0.19 | +0.08 | **+0.22** |
| $D_3$ | *0.72* | +0.03 | +0.25 | +0.09 | **+0.23** |
| $D_4$ | *0.69* | +0.06 | +0.22 | +0.08 | **+0.25** |
| $D_5$ | *0.74* | +0.08 | +0.19 | +0.04 | **+0.21** |
| Avg | *0.72* | +0.06 | + 0.20 | +0.07 | **+0.22** |

features considered by the system - by recording additional appropriate information during step a) above.

### *5.5.1 Experiments: Explicit Feature Extraction*

The previous review mining systems most relevant to our work are those in [49] and [126]. We only had access to the data used in [49] and the latter system is very difficult to reproduce, therefore our experiments include a comparison between OPINE and Hu&Liu's system, but no direct comparison between OPINE and IBM's SentimentAnalyzer [126] (see the related work section for a discussion of this work). Since Hu&Liu's system was widely considered the state-of-the-art system at the time we developed OPINE, we think that this comparison is sufficient.

Hu&Liu's system uses association rule mining to extract frequent review noun phrases as features. Frequent features are used to find potential opinion words (only adjectives) and the system uses WordNet synonyms and antonyms in conjunction with a set of seed words in order to find actual opinion words. Finally, infrequent phrases modified by opinion words discovered in this manner are extracted as well as product features. The system only extracts explicit features (as we will see later in the chapter, OPINE also extracts implicit features).

We compared OPINE with Hu&Liu's system using the 5 publicly available product electronics review sets introduced in [49] (500 reviews). We found that OPINE's precision is 22%

higher than Hu's at the cost of a 3% recall drop. There are two important differences between OPINE and Hu's system: a) OPINE's Feature Assessor uses PMI assessment to evaluate each candidate feature and b) OPINE incorporates Web PMI statistics in its assessment. In the following, we quantify the performance gains from a) and b).

a) In order to quantify the benefits of OPINE's Feature Assessor, we use the Assessor in conjunction with only the available review data in order to evaluate the potential features extracted by Hu's algorithm. The Feature Assessor improves Hu's precision by 6%.

b) In order to evaluate the impact of using Web PMI statistics, we assess OPINE's features first on reviews, and then on reviews in conjunction with the Web. Web PMI statistics increase precision by another 14.5% (on average) over the use of the review data alone.

Table 5.4: **Recall Comparison on the Explicit Feature Extraction Task.** OPINE's recall is 3% lower than the recall of Hu's original system (precision level = 0.8). All results are reported with respect to Hu's.

| **Dataset** | *Hu* | **Hu Assess(Reviews)** | **Hu Assess(Reviews,Web)** | **OPINE (Reviews)** | **OPINE** |
|---|---|---|---|---|---|
| $D_1$ | *0.82* | -0.16 | -0.08 | -0.14 | **-0.02** |
| $D_2$ | *0.79* | -0.17 | -0.09 | -0.13 | **-0.06** |
| $D_3$ | *0.76* | -0.12 | -0.08 | -0.15 | **-0.03** |
| $D_4$ | *0.82* | -0.19 | -0.04 | -0.17 | **-0.03** |
| $D_5$ | *0.80* | -0.16 | -0.06 | -0.12 | **-0.02** |
| Avg | *0.80* | -0.16 | -0.07 | -0.14 | **-0.03** |

Overall, 1/3 of **OPINE**'s precision increase over Hu's system comes from using PMI assessment on reviews and the other 2/3 from the use of the Web PMI statistics.

The drop in recall for our system is due to the fact that Hu's system proposes as features words or phrases that occur at least once in the review corpus, whereas OPINE requires a potential feature to occur at least twice. We ran OPINE with the same settings as Hu's system on a subset of the reviews (for 2 products) and the recall was higher than that of Hu's at higher precision; however, this is not a feasible setting for larger review sets - in order to be able to extract rare features, we are currently investigating methods that take advantage of sentence-level dependencies among product features; for example, once a feature $f$ has been identified, one can use enumerations (including conjunctions) or disjunctions to identify additional potential features that may only occur a small number of times in the given review corpus.

In order to show that OPINE's performance is robust across multiple product classes, we used two sets of 1307 reviews downloaded from `tripadvisor.com` for Hotels (1000 reviews) and `amazon.com` for Scanners (307 reviews) [1]. Two annotators labeled a set of unique 450 OPINE extractions as *correct* or *incorrect*. The inter-annotator agreement was 86%. The extractions on which the annotators agreed were used to compute OPINE's precision, which was 89%. Furthermore, the annotators extracted explicit features from 800 review sentences (400 for each domain). The inter-annotator agreement was 82%. OPINE's recall on the set of 179 features on which both annotators agreed was 73%. The explicit features for the Hotel domain can be seen as part of the OPINE demo accessible at *http://www.cs.washington.edu/homes/amp/opine/*; the explicit features for the Scanner domain can be found in Appendix C.1.

## 5.6 Finding Implicit Features

We now address the extraction of *implicit features*. The system first extracts *opinion phrases* attached to explicit features, as detailed in 5.7. Opinion phrases refer to *properties* (*e.g.*, "clean" refers to "cleanliness"). When the property is *implicit* (*e.g.*, "clean room"), the opinion is attached to an explicit feature (*e.g.*, "room"). OPINE examines opinion phrases associated with explicit features in order to extract implicit properties. If the opinion phrase is a verb, noun or adverb, OPINE associates it with `Quality`; if the opinion phrase is an adjective, OPINE maps it to a more specific property. For instance, if "clean" and "spacious" are opinions about hotel rooms, OPINE associates "clean" with `Cleanness` and "spacious" with `Size`.

The problem of associating adjectives with an implied property is closely related to that of finding adjectival scales [46]. OPINE uses WordNet synonymy and antonymy information

[1]See Appendix C.1 for information about how to obtain these review sets.

Table 5.5: **WordNet-based and Web-based Adjective Similarity Rules.** Notation: $s_1$, $s_2$ = WordNet synsets., *pertain*(), *attribute*() = relations between adjective and noun synsets.

| $adj_1$ **and** $adj_2$ **are similar if** |
|---|
| $\exists s_1, s_2$ s.t. $pertain(adj_1, s_1), attribute(adj_2, s_2), isA(s_1, s_2)$ |
| $\exists s_1, s_2$ s.t. $pertain(adj_1, s_1), pertain(adj_2, s_2), isA(s_1, s_2)$ |
| $\exists s_1, s_2$ s.t. $attribute(adj_1, s_1), attribute(adj_2, s_2), isA(s_1, s_2)$ |
| $\exists p \in \{$"$[X], even[Y]''$, "$[X], almost[Y]'', ...\}$ s.t. $hits(p(adj_1, adj_2)) > t, t = threshold$ |

to group the adjectives in a set of initial clusters. Next, any two clusters $A_1$ and $A_2$ are merged if multiple pairs of adjectives ($a_1$, $a_2$) exist such that $a_1 \in A_1$, $a_2 \in A_2$ and $a_1$ is *similar to* $a_2$ (an explanation of adjective similarity is given below). For example, $A_1$ = {"intuitive"} is merged with $A_2$ = {"understandable", "clear"}.

Clusters are labeled with the names of their corresponding properties (see Table 5.6). The property names are obtained from either WordNet (*e.g.*, *big* is a value of *size*), or from a name-generation module which adds suffixes (*e.g.*, "-iness", "-ity") to adjectives and uses the Web to filter out non-words and highly infrequent candidate names. If no property names can be found, the label is generated based on adjectives (*e.g.*, "beWelcome").

**Adjective Similarity** The adjective similarity rules in Table 5.5 consist of WordNet-based rules and Web-based rules. WordNet relationships such as $pertain(adjSynset, nounSynset)$ and $attribute(adjSynset, nounSynset)$ are used to relate adjectives to nouns representing properties: if two adjectives relate to the same property or to related properties, the two adjectives are similar. In addition to such WordNet-based rules, OPINE bootstraps a set of lexical patterns (see 5.8 for details) and instantiates them in order to generate search-engine queries which confirm that two adjectives correspond to the same property. Given clusters $A_1$ and $A_2$, OPINE instantiates patterns such as "$a_1$, (*) even $a_2$ " with $a_1 \in A_1$ and $a_2 \in A_2$ in order to check if $a_1$ and $a_2$ are similar. For example, hits *("clear, (*) even intuitive")* $> 5$, therefore "clear" is similar to "intuitive".

Given an explicit feature $f$ and a set of opinions associated with $f$ which have been clustered as previously described, OPINE uses the opinion clusters to extract *implicit features.* For example, given $f$=`Room` and opinions *clean*, *spotless* in the *Cleanness* cluster, OPINE

Table 5.6: **Examples of Labeled Opinion Clusters**

| | |
|---|---|
| **Quality** | *like, recommend, good, very good, incredibly good, great, truly great* |
| **Clarity** | *understandable, clear, straightforward, intuitive* |
| **Noise** | *quiet, silent, noisy, loud, deafening* |
| **Price** | *inexpensive, affordable, costly, expensive, cheap* |
| **Cleanness** | *clean, messy, dirty, very dirty, spotless* |

Table 5.7: **Domain-independent Rules for Potential Opinion Phrase Extraction.** Notation: po=potential opinion, M=modifier, NP=noun phrase, S=subject, P=predicate, O=object. Extracted phrases are enclosed in parentheses. Features are indicated by the typewriter font. The equality conditions on the left-hand side use *po*'s head.

| **Extraction Rules** | **Examples** |
|---|---|
| if $\exists(M, NP = f) \rightarrow po = M$ | (expensive) `scanner` |
| if $\exists(S = f, P, O) \rightarrow po = O$ | `lamp` has (problems) |
| if $\exists(S, P, O = f) \rightarrow po = P$ | I (hate) this `scanner` |
| if $\exists(S = f, P) \rightarrow po = P$ | `program` (crashed) |

generates the implicit feature `RoomCleanness`. We evaluated the impact of *implicit feature extraction* in the Hotels and Scanners domains [2]. Implicit features led to a 2% average increase in precision and a 6% increase in recall, mostly in the Hotel domain, which is rich in adjectives (*e.g.*, "*clean room*", "*soft bed*").

A list of found adjective clusters can be seen in Appendix C.1. The running time of the clustering algorithm is quadratic in the number of the extracted adjective opinions.

## 5.7 Finding Opinion Phrases and Their Polarity

This section describes how OPINE extracts potential opinion phrases, distinguishes between opinions and non-opinions, and finds the *polarity* of each opinion in the context of its associated feature in a particular review sentence.

OPINE uses explicit features to identify potential opinion phrases. Our intuition is that an opinion phrase associated with a product feature will occur in its vicinity. This idea is similar to that of [53] and [49], but instead of using a window of size $k$ or the output

[2]Hu's datasets have few implicit features and Hu's system doesn't handle implicit feature extraction.

Table 5.8: **Dependency Rule Templates For Finding Words $w$, $w'$ with Related Semantic Orientation Labels** Notation: v,w,w'=words; f, f'=feature names; dep=dependent; mod=modifier; conj = conjunction

| **Rule Templates** | **Example Rules** |
|---|---|
| $dep(w, w')$ | $mod(w, w')$ |
| $\exists v$ s.t. $dep(w, v), dep(v, w')$ | $\exists v$ s.t. $mod(w, v), object(v, w')$ |
| $\exists v$ s.t. $dep(w, v), dep(w', v)$ | $\exists v$ s.t. $mod(w, v), object(w', v)$ |
| $\exists f, f'$ s.t. $dep(w, f), dep(w', f'), dep(f, f')$ | $\exists f, f'$ s.t. $mod(w, f), mod(w', f'), conj(f, f')$ |

of a noun phrase chunker, OPINE takes advantage of the dependencies computed by the MINIPAR parser. Our intuition is embodied by a set of *extraction rules*, the most important of which are shown in Table 5.7. If an explicit feature is found in a sentence, OPINE applies the extraction rules in order to find the heads of potential opinion phrases. Each head word together with its modifiers is returned as a potential opinion phrase. The running time of extracting potential opinions is $O(|S|)$, where $|S|$ the number of review sentences.

OPINE examines the potential opinion phrases in order to identify the actual opinions. First, the system finds the semantic orientation for the lexical head of each potential opinion phrase. Every phrase whose head word has a *positive* or *negative* semantic orientation is then retained as an *opinion phrase*. In the following, we describe how OPINE finds the semantic orientation of words.

### *5.7.1 Context-specific Word Semantic Orientation*

Given a set of *semantic orientation (SO) labels* ($\{positive, negative, neutral\}$), a set of reviews and a set of tuples ($w$, $f$, $s$), where $w$ is a potential opinion word associated with feature $f$ in sentence $s$, OPINE assigns a SO label to each tuple ($w$, $f$, $s$). For example, the tuple (*sluggish*, *driver*, "I am not happy with this sluggish driver") will be assigned a *negative* SO label [3].

OPINE uses the 3-step approach below to label each ($w$, $f$, $s$) tuple:

1. Given the set of reviews, OPINE finds a SO label for each word $w$.

[3]We use "word" to refer to a potential opinion word $w$ and "feature" to refer to the word or phrase that represents the explicit feature $f$.

2. Given the set of reviews and the set of SO labels for words $w$, OPINE finds a SO label for each ($w$, $f$) pair.

3. Given the set of SO labels for ($w$, $f$) pairs, OPINE finds a SO label for each ($w$, $f$, $s$) input tuple.

Each of these subtasks is cast as an unsupervised collective classification problem and solved using the same mechanism. In each case, OPINE is given a set of *objects* (words, pairs or tuples) and a set of *labels* (SO labels); OPINE then searches for a *global* assignment of labels to objects. In each case, OPINE makes use of *local constraints* on label assignments (*e.g.*, conjunctions and disjunctions constraining the assignment of SO labels to words [47]).

A key insight in OPINE is that the problem of searching for a *global* SO label assignment to words, pairs or tuples while trying to satisfy as many *local* constraints on assignments as possible is analogous to labeling problems in computer vision (*e.g.*, model-based matching). OPINE uses a well-known computer vision technique, *relaxation labeling* [50], in order to solve the three subtasks described above.

### *5.7.2 Relaxation Labeling Overview*

Relaxation labeling is an iterative procedure that takes as input:

a) a set of *objects* (*e.g.*, words)

b) a set of *labels* (*e.g.*, SO labels)

c) initial probabilities for each object's possible labels

d) the definition of an object $o$'s *neighborhood* (a set of other objects which influence the choice of $o$'s label)

e) the definition of *neighborhood features*

f) the definition of a *support function* for an object label

The influence of an object $o$'s neighborhood on its label $L$ is quantified using the *support function*. The support function computes the probability of the label $L$ being assigned to $o$ as a function of $o$'s *neighborhood features*. Examples of features include the fact that a certain *local constraint* is satisfied (*e.g.*, the word *nice* participates in the conjunction *and* together with some other word whose SO label is estimated to be *positive*).

Relaxation labeling is an iterative procedure whose output is an assignment of labels to objects. At each iteration, the algorithm uses an *update equation* to re-estimate the probability of an object label based on its previous probability estimate and the features of its neighborhood. The algorithm stops when the global label assignment stays constant over multiple consecutive iterations.

We employ relaxation labeling for the following reasons: a) it has been extensively used, with good results, in computer vision research as well as in other projects, such as *ontology matching* [29]; b) its formalism allows for many types of constraints on label assignments to be used simultaneously. As mentioned before, constraints are integrated into the algorithm by means of neighborhood features that influence the assignment of a particular label to a particular object.

OPINE uses the following sources of constraints:

a) *conjunctions* (*e.g.*, "and") and *disjunctions* (*e.g.*, "but") in the review text.

b) a small number of manually-supplied *syntactic dependency rule templates* (see Table 5.8). The templates are automatically instantiated by our system with different dependency relationships (premodifier, postmodifier, etc.) in order to obtain syntactic dependency rules which find words with related SO labels.

c) automatically derived *morphological relationships* (*e.g.*, "wonderful" and "wonderfully" are likely to have similar SO labels).

d) WordNet-supplied *synonymy, antonymy, IS-A* and *morphological* relationships between words (the use of WordNet information is optional). For example, *clean* and *neat* are synonyms and so they are likely to have similar SO labels.

*Negation Modifiers* In order to correctly extract constraints on the SO labels of words (or, as we will later see, SO labels of (word,feature) and (word,feature,sentence) tuples), the system takes into consideration the presence of negation-type modifiers: *not*, *not at all*, *never*, *no*, *barely*, *hardly*, *scarcely*, etc. For the purposes of our experiments, we use a list of modifiers compiled from previous NLP literature; an item of current work is automatically discovering n-grams that function as negation modifiers as described in the following.

Starting with a) a set of seed words with known positive and negative semantic orientation labels and b) a set of high-precision constraints (such as the conjunction- and disjunction- based constraints), the system examines automatically extracted *unlikely* instances of the known constraints: a constraint instance is unlikely if the set of word labels it involves is dissimilar to those in other instances of the same constraint (in other words, if the constraint instance is an outlier). N-grams that are common across outlier constraint instances (after a normalization step) and across different constraint types can be extracted as potential negation modifiers.

Each of the SO label assignment subtasks previously identified is solved using a relaxation labeling step. In the following, we describe in detail how relaxation labeling is used to find SO labels for words in the given review sets.

### *5.7.3 Finding SO Labels for Words*

For many words, a word sense or set of senses is used throughout the review corpus with a consistently positive, negative or neutral connotation (*e.g.*, “great”, “awful”, etc.). Thus, in many cases, a word $w$’s SO label in the context of a feature $f$ and sentence $s$ will be the same as its SO label in the context of other features and sentences. In the following, we describe how OPINE’s relaxation labeling mechanism is used to find a word’s dominant SO label in a set of reviews.

For this task, a word’s *neighborhood* is defined as the set of words connected to it through conjunctions, disjunctions and all other relationships previously introduced as sources of constraints.

RL uses an *update equation* to re-estimate the probability of a word label based on its previous probability estimate and the features of its neighborhood (see **Neighborhood Features**). At iteration $m$, let $q(w, L)_{(m)}$ denote the support function for label $L$ of $w$ and let $P(l(w) = L)_{(m)}$ denote the probability that $L$ is the label of $w$. $P(l(w) = L)_{(m+1)}$ is computed as follows:

*General RL Update Equation* [94]

$$P(l(w) = L)_{(m+1)} = \frac{P(l(w) = L)_{(m)}(1 + \alpha q(w, L)_{(m)})}{\sum_{L'} P(l(w) = L')_{(m)}(1 + \alpha q(w, L')_{(m)})}$$

where $L' \in \{pos, neg, neutral\}$ and $\alpha > 0$ is an experimentally set constant. This form of the update equation adds 1 to the support function and includes an $\alpha$ factor in order to ensure that the computed value is positive under any circumstances (e.g., if the used support function has negative values). The values of the main support function OPINE used in its experiments are positive values in the [0,1] interval - other support functions with which we experimented also had negative values. If the values of the support function are known to be positive, the normalization factors from the above equation can be omitted.

RL's output is an assignment of dominant SO labels to words.

In the following, we describe in detail the initialization step, the support function and the use of neighborhood features.

**RL Initialization Step** OPINE uses a version of Turney's PMI-based approach [113] in order to derive the initial probability estimates ($P(l(w) = L)_{(0)}$) for a subset $S$ of the potential opinion words (since the process of getting the necessary hitcounts can be expensive, $S$ contains either the top $k$% most frequent potential opinions or the potential opinions that are linked to the largest numbers of other words by means of constraint-inducing dependencies (in other words, the "best connected" potential opinions)). The experimental results in this chapter use the first definition of $S$ and $k = 15\%$ ($k = 20\%$ led to similar results). In the future, we plan to experiment with the use of large, domain-independent opinion lexicons (currently being built by a number of researchers, the author included) as part of the initialization step.

OPINE computes a *SO score so(w)* for each $w$ in $S$ as the difference between the PMI of $w$ with positive keywords (*e.g.*, "excellent") and the PMI of $w$ with negative keywords (*e.g.*, "awful"). When $so(w)$ is small, or $w$ rarely co-occurs with the keywords, $w$ is classified as *neutral*. Otherwise, if $so(w) > 0$ $w$ is *positive*, and if $so(w) < 0$ $w$ is *negative*. OPINE then uses the labeled $S$ set in order to compute prior probabilities $P(l(w) = L)$, $L \in \{pos, neg, neutral\}$ by computing the ratio between the number of words in $S$ labeled $L$ and $|S|$. These probabilities will be used as initial probability estimates associated with the labels of the words outside of $S$.

**Support Function** The support function computes the probability of each label for word $w$ based on the labels of objects in $w$'s neighborhood $N$.

Let $A_k = \{(w_j, L_j)|w_j \in N\}$ , $0 < k \leq 3^{|N|}$ represent one of the potential assignments of labels to the words in $N$. Let $P(A_k)_{(m)}$ denote the probability of this particular assignment at iteration $m$. The *support* for label $L$ of word $w$ at iteration $m$ is :

$$q(w, L)_{(m)} = \sum_{k=1}^{3^{|N|}} P(l(w) = L|A_k)_{(m)} * P(A_k)_{(m)}$$

We assume that the labels of $w$'s neighbors are independent of each other and so the formula becomes:

$$q(w, L)_{(m)} = \sum_{k=1}^{3^{|N|}} P(l(w) = L|A_k)_{(m)} * \prod_{j=1}^{|N|} P(l(w_j) = L_j)_{(m)}$$

Every $P(l(w_j) = L_j)_{(m)}$ term is the estimate for the probability that $l(w_j) = L_j$ (which was computed at iteration $m$ using the RL update equation).

The $P(l(w) = L|A_k)_{(m)}$ term quantifies the influence of a particular label assignment to $w$'s neighborhood over $w$'s label. In the following, we describe how we estimate this term.

**Neighborhood Features** Each type of word relationship which constrains the assignment of SO labels to words (synonymy, antonymy, conjunction, morphological relations etc.) is mapped by OPINE to a neighborhood feature. This mapping allows OPINE to simultaneously use multiple independent sources of constraints on the label of a particular word. In the following, we formalize this mapping.

Let $T$ denote the type of a word relationship in $R$ and let $A_{k,T}$ represent the labels assigned by $A_k$ to neighbors of a word $w$ which are connected to $w$ through a relationship of type $T$ . We have $A_k = \bigcup_T A_{k,T}$ and

$$P(l(w) = L|A_k)_{(m)} = P(l(w) = L|\bigcup_T A_{k,T})_{(m)}$$

For each relationship type $T$, OPINE defines a *neighborhood feature* $f_T(w, L, A_{k,T})$ which computes $P(l(w) = L|A_{k,T})$, the probability that $w$'s label is $L$ given $A_{k,T}$ (see below). $P(l(w) = L|\bigcup_T A_{k,T})_{(m)}$ is then estimated combining the information from various features about $w$'s label as follows:

$$P(l(w) = L|A_k)_{(m)} = \sum_{i=1}^{j} f_i(w, L, A_{k,i})_{(m)} * c_i$$

where $c_0, ...c_j$ are weights whose sum is 1 and which reflect OPINE 's confidence in each type of feature. An alternative means of computing this quantity uses the sigmoid function, which has previously been shown to work well in a similar context but in a different application (ontology matching).

Given word $w$, label $L$, relationship type $T$ and neighborhood label assignment $A_k$, let $N_T$ represent the subset of $w$'s neighbors connected to $w$ through a type $T$ relationship. The feature $f_T$ computes the probability that $w$'s label is $L$ given the labels assigned by $A_k$ to words in $N_T$:

$$f_T(w, L, A_{k,T})_{(m)} = P(l(w) = L|L_i, ...L_n),$$

where $L_i, ...L_n$ represent the labels of the neighbors in $N_T$.

This probability can be evaluated in a number of ways, such as:

a)

$$f_T(w, L, A_{k,T})_{(m)} = \frac{\sum_{j=1}^{|N_T|} P(l(w) = L|L_j)}{|N_T|}$$

or

b) using Bayes's Law and making an independence assumption:

$$f_T(w, L, A_{k,T})_{(m)} = \frac{P(l(w) = L)_{(m)} * \prod_{j=1}^{|N_T|} P(L_j|l(w) = L)}{P(A_{k,T})_{(m)}}.$$

$P(L_j|l(w) = L)$ is the probability that word $w_j$ has label $L_j$ if $w_j$ and $w$ are linked by a relationship of type $T$ and $w$ has label $L$. We make the following assumptions, which are appropriate in our case and supported by both the constraint types involved and by data: I) $P(l(w) = L|L_j)$ depends only on $T$, $L$ and $L_j$, not of the particular words $w_j$ and $w$ and II) $P(L_j|l(w) = L) = P(l(w) = L|L_j)$.

Table 5.9: **Examples of Conditional Probabilities for Neighboring Words or Tuples Linked by Conjunctions or Disjunctions.** Notation: $p(l(e) = L|link, l(e') = L')$ = probability that entity $e$ (word or tuple) has label $L$ given the current label $L'$ of $e'$ and the type of link between $e$ and $e'$; $(conj, +)$ = link type is *conj* and label of $e'$ is $+$, $(conj, |)$ = link type is *conj* and label of $e'$ is $|$ and so on. $+$ = positive label, $-$ = negative label, $|$ = neutral label.

| $p(l(e) = L\|link, l(e') = L')$ | $conj, +$ | $conj, -$ | $conj, \|$ | $disj, +$ | $disj, -$ | $disj, \|$ |
|---|---|---|---|---|---|---|
| $l(e) = +$ | 0.8 | 0.08 | 0.12 | 0.09 | 0.79 | 0.12 |
| $l(e) = -$ | 0.08 | 0.78 | 0.14 | 0.79 | 0.09 | 0.12 |
| $l(e) = \|$ | 0.12 | 0.14 | 0.74 | 0.12 | 0.12 | 0.76 |

For each tuple $(T,\ L,\ L_j)$, $L, L_j \in \{pos, neg, neutral\}$, OPINE uses a probability table built using a small set of positive, negative and neutral words (see Table 5.9 for some examples).

### *5.7.4 Finding (Word, Feature) SO Labels*

This subtask is motivated by the existence of frequent words which change their SO label based on associated features, but whose SO labels in the context of the respective features are consistent throughout the reviews (*e.g.*, in the Hotel domain, “hot water” has a consistently positive connotation, whereas “hot room” has a negative one).

In order to solve this task, OPINE initially assigns each $(w, f)$ pair the (label, probability) information corresponding to $w$. The system then executes a relaxation labeling step during which syntactic relationships between words and, respectively, between features, are used to update the default SO labels whenever necessary. For example, *(hot, room)* appears in the proximity of *(broken, fan)*. If “room”and “fan” are conjoined by *and*, this suggests that “hot” and “broken” have similar SO labels in the context of their respective features. If “broken” has a strongly negative semantic orientation, this fact contributes to OPINE’s belief that “hot” may also be negative in this context. Since *(hot, room)* occurs in the vicinity of other such phrases (*e.g.*, *stifling kitchen*), “hot” acquires a negative SO label in the context of “room”.

### *5.7.5 Finding (Word, Feature, Sentence) SO Labels*

This subtask is motivated by the existence of $(w,f)$ pairs (*e.g.*, *(big, room)*) for which $w$'s orientation changes depending on the sentence in which the pair appears (*e.g.*, " I hated the big, drafty room because I ended up freezing" vs. "We had a big, luxurious room".)

In order to solve this subtask, OPINE first initializes the label probabilities for each $(w, f, s)$ tuple with the label probabilities for the $(w, f)$ pair. The system then uses syntactic relationships between words and, respectively, features in order to update the SO labels when necessary. For example, in the sentence "I hated the big, drafty room because I ended up freezing.", "big" and "hate" satisfy condition 2 in Table 5.8 and therefore OPINE expects them to have similar SO labels. Since "hate" and "drafty" have strong negative connotations, "big" acquires a negative SO label in this context.

### *5.7.6 Identifying Opinion Phrases*

After OPINE has computed the most likely SO labels for the head words of each potential opinion phrase in the context of given features and sentences, OPINE can extract *opinion phrases* ("very hot", "incredibly beautiful", "etc.) and establish their polarity. Phrases whose head words have been assigned *positive* or *negative* labels are retained as *opinion phrases*. Furthermore, the polarity of an opinion phrase $o$ in the context of a feature $f$ and sentence $s$ is given by the SO label assigned to the tuple $(head(o), f, s)$ (the system takes into account the presence of negation modifiers when appropriate).

### *5.7.7 Experiments*

In this section we evaluate OPINE's performance on the following tasks: finding SO labels of words in the context of known features and sentences (*word SO label extraction*); distinguishing between opinion and non-opinion phrases in the context of known features and sentences (*opinion phrase extraction*); finding the correct polarity of extracted opinion phrases in the context of known features and sentences (*opinion phrase polarity extraction*).

We first ran OPINE on 13841 sentences and 538 previously extracted features; as in the case of explicit features extraction, we used the product electronics data in Hu&Liu's

Table 5.10: **Finding Word Semantic Orientation Labels in the Context of Given Features and Sentences.** OPINE's precision is higher than that of **PMI++** and **Hu++**. All results are reported with respect to **PMI++**.

| **Word POS** | **PMI++** | | **Hu++** | | **OPINE** | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **Precision** | **Recall** | **Precision** | **Recall** |
| Adjectives | 0.73 | 0.91 | +0.02 | -0.17 | **+0.07** | **-0.03** |
| Nouns | 0.63 | 0.92 | +0.04 | -0.24 | **+0.11** | **-0.08** |
| Verbs | 0.71 | 0.88 | +0.03 | -0.12 | **+0.01** | **-0.01** |
| Adverbs | 0.82 | 0.92 | +0.02 | -0.01 | **+0.06** | **+0.01** |
| Avg | 0.72 | 0.91 | +0.03 | -0.14 | **+0.06** | **-0.03** |

experiments, as well as review data from the Hotel and Scanner domains.

OPINE searched for a most likely SO label assignment for 1756 different words in the context of the given features and sentences. We compared OPINE against two baseline methods, **PMI++** and **Hu++**.

**PMI++** is an extended version of [116]'s method for finding the SO label of a word or a phrase. For a given (word, feature, sentence) tuple, **PMI++** ignores the sentence, generates a phrase containing the word and the feature (*e.g.*, "clean room") and finds its SO label using PMI statistics. If unsure of the label, **PMI++** finds the orientation of the potential opinion word instead. The search engine queries use domain-specific keywords (*e.g.*, "clean room" + "hotel"), which are dropped if they lead to low counts. **PMI++** also uses morphology information (*e.g.*, *wonderful* and *wonderfully* are likely to have similar semantic orientation labels).

**Hu++** is a WordNet-based method for finding a word's context-independent semantic orientation. It extends Hu's adjective labeling method [49] in order to handle nouns, verbs and adverbs and in order to improve coverage. Hu's method starts with two sets of positive and negative words and iteratively grows each one by including synonyms and antonyms from WordNet. The final sets are used to predict the orientation of an incoming word. **Hu++** also makes use of WordNet IS-A relationships (*e.g.*, $problem_0$ IS-A $difficulty_0$) and morphology information.

### *5.7.8 Experiments: Word SO Labels*

On the task of finding SO labels for words in the context of given features and review sentences, OPINE obtains higher precision than both baseline methods at a small loss in recall with respect to **PMI++**. As described below, this result is due in large part to OPINE's ability to handle context-sensitive opinion words.

We randomly selected 200 (word, feature, sentence) tuples for each word type (adjective, adverb, etc.) and obtained a test set containing 800 tuples. The sources of these tuples are reviews from all the available product domains: the Hotel reviews contain predominantly adjective opinions (and some noun opinions), while product electronics reviews (including Scanner reviews) contain primarily adjective as well as verb and adverb opinions.

Two annotators assigned positive, negative and neutral labels to each tuple (the inter-annotator agreement was 78%). We retained the tuples on which the annotators agreed as the gold standard. We ran **PMI++** and **Hu++** on the test data and compared the results against OPINE's results on the same data.

In order to quantify the benefits of each of the three steps of our method for finding SO labels, we also compared OPINE with a version which only finds SO labels for words and a version which finds SO labels for words in the context of given features, but doesn't take into account given sentences. We have learned from this comparison that OPINE's precision gain over **PMI++** and **Hu++** is mostly due to its ability to handle context-sensitive words in a large number of cases.

Although **Hu++** does not handle context-sensitive SO label assignment, its average precision was reasonable (75%) and better than that of **PMI++**. Finding a word's SO label is good enough in the case of strongly positive or negative opinion words, which account for a large number of opinion instances. However, the method's recall is limited due to a number of reasons: it is unable to handle words or phrases absent from WordNet (*e.g.*, "depth-adjustable"); it refuses to label highly polysemous words whose semantic orientation depends on the word sense; it also refuses to label WordNet terms that cannot be "reached" from the seed set of positive/negative words by means of WordNet relations (synonymy, antonymy, IS-A).

**PMI++** typically does well in the presence of strongly positive or strongly negative words. Its main shortcoming is misclassifying terms such as "casual" or "basic" that change orientation based on context (see Table 5.11 for examples of such words and see Appendix C.1 for examples of specific contexts - features and sentences).

Appendix C.1 also contains examples of positive or negative dominant semantic orientation labels assigned by our system to words in the Hotel review set.

Table 5.11: **Context-sensitive Words**

| | | | |
|---|---|---|---|
| open | closed | young | old |
| new | full | empty | big |
| small | central | straight | deep |
| atypical | downtown | uptown | homemade |
| low | high | real | common |
| solid | bound | dark | simple |
| soft | formal | casual | informal |
| joint | separate | thin | thick |

### *5.7.9 Experiments: Opinion Phrases*

In order to evaluate OPINE on the tasks of *opinion phrase extraction* and *opinion phrase polarity extraction* in the context of known features and sentences, we used a set of 550 sentences containing previously extracted features. The sentences were annotated with the opinion phrases corresponding to the known features and with the opinion polarity. The task of *opinion phrase polarity extraction* differs from the task of *word SO label assignment*

Table 5.12: **Extracting Opinion Phrases and Opinion Phrase Polarity In the Context of Known Features and Sentences.** OPINE's precision is higher than that of **PMI++** and **Hu++**. All results are reported with respect to **PMI++**.

| **Measure** | **PMI++** | **Hu++** | **OPINE** |
|---|---|---|---|
| Opinion Extraction: Precision | 0.71 | +0.06 | **+0.08** |
| Opinion Extraction: Recall | 0.78 | -0.08 | **-0.02** |
| Opinion Polarity: Precision | 0.80 | -0.04 | **+0.06** |
| Opinion Polarity: Recall | 0.93 | +0.07 | **-0.04** |

above as follows: the polarity extraction for opinion phrases only examines the assignment of *pos* and *neg* labels to phrases which were found to be opinions (that is, not *neutral*) after the *word SO label assignment* stage is completed.

We compared OPINE with **PMI++** and **Hu++** on the tasks of interest. We found that OPINE had the highest precision on both tasks at a small loss in recall with respect to **PMI++**. OPINE's ability to identify a word's SO label in the context of a given feature and sentence allows the system to correctly extract opinions expressed by words such as "big" or "small", whose semantic orientation varies based on context.

*Challenges*

OPINE's performance is negatively affected by a number of factors: parsing errors lead to missed candidate opinions and incorrect opinion polarity assignments; other problems include sparse data (in the case of infrequent opinion words) and complicated opinion expressions (*e.g.*, nested opinions, conditionals, subjunctive expressions). In our experience, some knowledge domains have been easier to handle than others: product reviews contained more colloquial language as well as more complicated opinion expressions, while the Hotel reviews were better written (containing simpler, more concise and grammatical sentences, etc.). However, this is a likely artifact of data collection from different sites with different user profiles; in the future, we plan to build on preliminary work on automatically identifying *helfpul* reviews (reviews that are easy to understand while providing information about salient product features).

## *5.8 Ranking Opinion Phrases*

OPINE clusters opinions in order to identify the properties to which they refer. Given an opinion cluster $A$ corresponding to some property, OPINE ranks its elements based on their *relative strength*. The probabilities computed at the end of the relaxation-labeling scheme generate an initial opinion ranking.

In order to improve this initial ranking, OPINE uses additional Web-derived constraints on the relative strength of phrases. As pointed out in [46], patterns such as "$a_1$, (*) even $a_2$" are good indicators of how strong $a_1$ is relative to $a_2$. To our knowledge, the sparse data problem mentioned in [46] has so far prevented such strength information from being

Table 5.13: **Lexical Patterns Used to Derive Opinions' Relative Strength.**

| | |
|---|---|
| $a, (*)$ *even* $b$ | $a, (*)$ *not* $b$ |
| $a, (*)$ *virtually* $b$ | $a, (*)$ *almost* $b$ |
| $a, (*)$ *near* $b$ | $a, (*)$ *close to* $b$ |
| $a, (*)$ *quite* $b$ | $a, (*)$ *mostly* $b$ |

computed for adjectives from typical news corpora. However, the Web allows us to use such patterns in order to refine our opinion rankings. OPINE starts with the pattern mentioned before and bootstraps a set of similar patterns (see Table 5.13). Given a cluster $A$, queries which instantiate such patterns with pairs of cluster elements are used to derive constraints such as:

$c_1 = (strength(deafening) > strength(loud))$,

$c_2 = (strength(spotless) > strength(clean))$.

OPINE also uses synonymy and antonymy-based constraints, since synonyms and antonyms tend to have similar strength:

$c_3 = (strength(clean) = strength(dirty))$.

The set $S$ of such constraints induces a constraint satisfaction problem (CSP) whose solution is a ranking of the cluster elements affected by $S$ (the remaining elements maintain their default ranking). In the general case, each constraint would be assigned a probability $p(s)$ and OPINE would solve a probabilistic CSP as described in [36]. We simplify the problem by only using constraints supported by multiple patterns in Table 5.13 and by treating them as hard rather than soft constraints. Finding a strength-based ranking of cluster adjectives amounts to a topological sort of the induced constraint graph. In addition to the main opinion word, opinion phrases may contain *intensifiers* (*e.g.*, *very*). The patterns in Table 5.13 are used to compare the strength of modifiers (*e.g.*, $strength(very) > strength(somewhat)$) and modifiers which can be compared in this fashion are retained as *intensifiers.* OPINE uses intensifier rankings to complete the adjective opinion rankings (*e.g.*, "very nice" is stronger than "somewhat nice"). In order to assess OPINE's performance on the opinion ranking task, we scored the set of adjective opinion rankings for the top 20 most frequent properties as follows: every partial ranking of the type $strength(op) > strength(op')$ produced by the system (where *op* and $op'$ are opinions), is scored as correct or incorrect by a human judge. The system's precision was 73%.

## 5.9 *Identifying and Analyzing Opinion Sentences*

In order to fully compare OPINE with the related system in [49], we look at how to extract *opinion sentences* and their polarity. [49] extracts sentences containing a product feature and at least one opinion word (an adjective) as *opinion sentences.* OPINE refines this definition by extracting each sentence that contains at least one product class feature and at least one *corresponding* opinion phrase (adjective, adverb, verb or noun) as an *opinion sentence.* After opinion sentences are extracted, we classify them as positive or negative.

For each feature $f$ with associated opinion phrases $op_i, ...op_j$ in sentence $s$, OPINE uses the harmonic mean formula to derive the probability that the *overall opinion label* for $f$ is $L$, where $L \in \{pos, neg\}$ and $k$ is the number of opinion phrases with label $L$:

$$P(L|f) = k * \frac{\prod_{i=0}^{k} strength(op_i)}{\sum_{i=0}^{k} strength(op_i)}$$

If $P(pos|f) > P(neg|f)$, the *overall opinion* associated with $f$ is positive, otherwise it is negative.

In order to find the overall opinion label associated with the sentence, OPINE uses the same formula to combine the information from the features with positive and, respectively, negative labels:

$$P(L|s) = m * \frac{\prod_{i=0}^{m} P(L|f_i)}{\sum_{i=0}^{m} P(L|f_i)}$$

where $L \in \{pos, neg\}$ and $m$ is the number of sentence features with overall opinion label $L$.

If $|P(pos|s) - P(neg|s)| > t$ (t = experimentally set threshold, OPINE classifies the sentence as positive if $P(pos|s) > P(neg|s)$ and negative otherwise. If $|P(pos|s) - P(neg|s)| < t$, OPINE uses the fact that same-polarity sentences tend to appear in the same paragraph and assigns to the sentence $s$ the polarity of the previous sentence in the review. Table 5.14 contains examples of positive and negative sentences, together with an example of a sentence whose polarity is decided by that of the previous review sentence.

Table 5.14: **Examples of Opinion Sentences and their Polarity**. Notation: "+" = positive, "-" = negative, "?" = undecided (using polarity of previous sentence). The italic font indicates opinion phrases and the typewriter font indicates product features.

| **Polarity** | **Sentence** |
|---|---|
| + | `It` was *immaculate, comfortable* (`beds` are a *dream*), *centrally located* and `the service` was *amazing.* |
| + | I would highly *recommend* this `hotel`. |
| + | We *enjoyed* our `breakfasts`, what a great way to start the day. |
| - | `Prices` are *competitive*, but `it` can be *difficult* to book. |
| - | We stayed in the Shubert suite and `it` was *noisy* late into the night. |
| ? | Our `room` was *clean*, but when we walked into the `bathroom`, we saw `it` was *dirty.* |

### *5.9.1 Experimental Results: Opinion Sentences*

In order to quantify the advantage of using OPINE's features and corresponding opinions, we compared OPINE and Hu's system on the tasks of extracting opinion sentences and determining their polarity. On the first task, Hu has 64% precision and 69% recall. OPINE obtains 86% precision and 80% recall, which corresponds to 16% F-measure increase (see Table 5.15).

In order to quantify the advantage of using OPINE's high-quality features, we compared Hu's original system (*Hu*) with a version of OPINE which uses OPINE's explicit features, but keeps the other characteristics of Hu's system intact (Hu's opinion sentence definition, Hu's absence of a pronoun resolution module and Hu's exclusive use of adjectives as potential opinions). This version of OPINE, **OP(F)**, registers a 4% F-measure increase over Hu's system.

In order to quantify the relative advantage of OPINE's additions and changes with respect to Hu's system, we compare **OP(F)** to three other versions of OPINE : **OP(F,Def)** incorporates OPINE's opinion sentence definition (Def), **OP(F,Def,Pron)** incorporates simple

Table 5.15: **Opinion Sentence Extraction Comparison (F-measure values)**. OPINE outperforms Hu's original system by 16% (22% on precision, 11% on recall). The improvement is due to OPINE's better features, better sentence definition, use of a simple pronoun resolution module and augmented set of opinion types. All improvements are reported with respect to Hu's system. Notation: **OP(F)** = method similar to *Hu*, but using OPINE's features, **OP(F,Def)** = method similar to **OP(F)**, but using OPINE's definition of an opinion sentence, **OP(F,Def,Pron)** = method similar to **OP(F,Def)**, but using a simple pronoun resolution module.

| **Dataset** | **Opinion Sentence Extraction: Hu vs. OPINE** | | | | |
|---|---|---|---|---|---|
| | *Hu* | **OP(F)** | **OP(F,Def)** | **OP(F,Def,Pron)** | **OPINE** |
| $D_1$ | *0.67* | +0.04 | +0.11 | +0.13 | **+0.16** |
| $D_2$ | *0.58* | +0.06 | +0.13 | +0.16 | **+0.24** |
| $D_3$ | *0.73* | +0.04 | +0.07 | +0.09 | **+0.11** |
| $D_4$ | *0.67* | +0.05 | +0.08 | +0.11 | **+0.14** |
| $D_5$ | *0.62* | +0.03 | +0.07 | +0.11 | **+0.15** |
| Avg | *0.66* | +0.04 | +0.09 | +0.12 | **+0.16** |

pronoun resolution and finally, **OPINE** adds additional opinion types (nouns, verbs, adverbs). Comparing **OP(F,Def)** to **OP(F)** shows that a more restrictive opinion sentence definition in conjunction with high-precision feature and opinion phrase extraction leads to increased performance.

Comparing **OP(F,Def,PronRes)** to **OP(F,Def)** shows that simple pronoun resolution helps as well. Finally, comparing **OPINE** to **OP(F,Def,PronRes)** shows the benefits of including nouns, verbs and adverbs in the opinion vocabulary. The datasets used in [49] consist of consumer electronics reviews; while the vast majority of opinion phrases in hotel or restaurant reviews are adjective phrases, consumer electronics contain a large number of verb and adverb phrases describing people's opinions about a product's functionality ("works", "breaks", etc.). Therefore, OPINE benefits from being able to handle multiple opinion types.

As seen in Table 5.16, on the task of determining opinion sentence polarity, OPINE sees an 8% increase over the 84% accuracy of Hu's system. OPINE benefits from including adverbs, nouns and verbs in the set of possible opinions.

Table 5.16: **Sentence Polarity Extraction Comparison.** OPINE's accuracy is 8% higher than that of Hu's. The differences between OPINE's results and Hu's are in bold.

| **Dataset** | **Hu** | **OPINE** |
|---|---|---|
| $D_1$ | 0.92 | **+0.01** |
| $D_2$ | 0.95 | **-0.03** |
| $D_3$ | 0.76 | **+0.12** |
| $D_4$ | 0.84 | **+0.11** |
| $D_5$ | 0.73 | **+0.18** |
| *Average* | 0.84 | **+0.08** |

## *5.10 Related Work*

The review-mining work most relevant to our research is described in [49], [54] and [126]. All three systems identify product features from reviews, but OPINE significantly improves on the first two and its reported precision is comparable to that of the third (although we were not able to perform a direct comparison, as the system and the data sets are not available). [49] doesn't assess candidate features, so its precision is lower than OPINE's. [54] employs an iterative semi-automatic approach which requires human input at every iteration. Neither model explicitly addresses *composite* (feature of feature) or *implicit* features. [126] uses a sophisticated feature extraction algorithm whose precision is comparable to OPINE's much simpler approach; OPINE's use of meronymy lexico-syntactic patterns is inspired by papers such as [9] and [4]. Other systems [74, 56] also look at Web product reviews but they do not extract opinions about particular product features.

Recognizing the subjective character and polarity of words, phrases or sentences has been addressed by many authors, including [113, 97, 47]. Most recently, [111] reports on the use of spin models to infer the semantic orientation of words. The chapter's global optimization approach and use of multiple sources of constraints on a word's semantic orientation is similar to ours, but the mechanism differs and the described approach omits the use of syntactic information. Subjective phrases are used by [116, 81, 56, 53] and others in order to classify reviews or sentences as positive or negative. So far, OPINE's focus has been on extracting and analyzing opinion phrases corresponding to specific features in specific sentences, rather than on determining sentence or review polarity. To our knowledge, [126] and [122] describe the only other systems which address the problem of finding context-specific word semantic orientation. [126] uses a large set of human-generated patterns which determine the final semantic orientation of a word (in the context of a product feature) given its prior semantic orientation provided by an initially supplied word list. OPINE's approach, while independently developed, amounts to a more general version of the approach taken by [126]: OPINE automatically computes both the prior and final word semantic orientation using a relaxation labeling scheme which accommodates multiple constraints. [122] uses a supervised approach incorporating a large set of features in order to learn the types of linguistic contexts which alter a word's prior semantic orientation. The paper's task is different than the one addressed by OPINE and [126], as it involves open-domain text and lacks any information about the target of a particular opinion.

[113] suggests using the magnitude of the PMI-based SO score as an indicator of the opinion's strength while [123, 40] use a supervised approach with large lexical and syntactic feature sets in order to distinguish among a few strength levels for sentence clauses. OPINE's unsupervised approach combines Turney's suggestion with a set of strong ranking constraints in order to derive opinion phrase rankings.

Finding properties implied by adjective opinions in reviews is related to finding adjectival scales [46]. OPINE benefits from the use of Web- and WordNet-derived information in clustering adjective opinions; also, the space of properties to which reviews refer tends to be smaller than in the case of a large news corpus.

## 5.11 Conclusions and Future Work

OPINE is an unsupervised information extraction system which extracts fine-grained features, and associated opinions, from reviews. OPINE's use of the Web as a corpus helps identify product features with improved precision compared with previous work. OPINE uses a novel relaxation-labeling technique to determine the semantic orientation of potential opinion words in the context of the extracted product features and specific review sentences; this technique allows the system to identify customer opinions and their polarity with high precision and recall. In the future, we plan to extend OPINE's techniques to open-domain text.

# Chapter 6
# ACQUIRING COMMONSENSE INFORMATION FROM WEB TEXT

## *6.1 Introduction*

In the past few years advances in probabilistic inference and sensor technology have led to a new generation of integrated AI systems that includes systems for activity and state recognition. Such systems have been found to benefit from commonsense information about daily human activities available in volunteer-created knowledge bases such as OpenMind and its offshoot, the Open Mind database for Indoor Common Sense (OMICS). However, knowledge bases created by volunteers suffer from semantic gaps and noise - additionally, the human effort involved in creating and updating them is significant. In this chapter, we show how Web data can be used to clean up and augment existent commonsense knowledge bases for integrated AI systems.

The contributions of this chapter are as follows:

1. We describe a simple, high precision method that uses Web-scale statistics to assess potential event-level relation instances and show that the assessed data significantly improves the precision and accuracy of a state-of-the-art system for state recognition [86].

2. We show that *focused mining* of event-level relation instances from the Web can mitigate the knowledge gaps in volunteer-created commonsense knowledge bases by discovering *new* high quality facts. We then evaluate the mined data in the context of the state recognition task and show that it leads to results comparable to those obtained when using the assessed version of volunteer-supplied data from 1. In other words, volunteer created content can be replaced by automatically mined information (in preparation).

The rest of this chapter is organized as follows: Section 6.2 describes the domain model for household activities and introduces the OMICS knowledge base created for integrated AI systems; Section 6.3 shows how instances of event-level commonsense relations can be assessed using Web-scale statistics; Section 6.4 describes the iterative acquisition of relation instances with the help of Web data ; Section 6.5 gives an overview of our results; Section 6.6 discusses related work areas and Section 6.7 gives an overview of our on-going work.

## *6.2 A Domain Model for Household Activities*

This section describes the structure of a basic domain model for commonsense information specific to household activities. A subset of the domain-specific predicates can be populated using techniques described in previous chapters of this thesis, as well as other NLP methods (see subsection 6.2.3) - for the purpose of this chapter, these predicates will be treated as *background knowledge*. The remaining predicates are *complex target predicates* (see subsection 6.2.2); the current chapter focuses on automatically populating and assessing them using Web data.

### *6.2.1 Domain Model for Household Activities: Background Knowledge*

Given a set of *concepts* (*e.g.*, *Person, Table*), their corresponding *properties* (*e.g.*, $TableCleanliness$) and a set of *relations* among *concepts* (*e.g.*, $Person, Clean, Table$), we are interested in *state* and *action events* defined as follows:

a) A *state* event is a tuple $e$ = *[concept, Be, propertyValue]*, where *Be* denotes the IS-A relation and *propertyValue* denotes the value of a concept property.

For example, $e$ = *[Door, Be, Clean]* corresponds to a particular state of the door. In the following, we omit the relation and simply denote each state by a *[concept, propertyValue]* tuple.

b) An *action* event is a tuple *[concept, relation, conceptSet]*.

For example, $e$ = *[Person, Open, Door]* corresponds to a particular *action* performed by Person and involving a Door object. The concepts in the *conceptSet* correspond to direct or indirect objects: for example, another potential complex action event is $e$ = *[Person, Open, Door, (with) Hand]*. In this case, the indirect object *Hand* is preceded by the corresponding preposition *with*.

Given a set of state and action events, we define the following utility predicates that will be used to characterize higher-level target predicates.

a) $sameImplicitProperty(e_0, e_1)$: the value of this predicate is True if $e_0$ and $e_1$ are states corresponding to the same implicit property (*e.g.*, *[Door, Open]*, *[Door, Closed]*) and False under any other circumstances. Given an event pair, the value of this predicate is computed automatically using the available domain model.

b) $positiveSemanticOrientation(e)$, $negativeSemanticOrientation(e)$, $neutralSemanticOrientation(e)$ are predicates that characterize a particular event with respect to its desirability: for example, $[Door, Broken]$ has a negative semantic orientation. These predicates were evaluated using a version of Turney's PMI-IR method as described in Chapter 5 - we are also able to use our own techniques from Chapter 5 to evaluate the semantic orientation of a particular event in the context of another (*e.g.*, $[Water, Hot]$ has a positive connotation in the context of $[Person, Make, Tea]$, but not in the context of $[Person, Water, Plant]$), but the driving state recognition application described in Section 6.5 has yet to incorporate this type of information in its model.

### *6.2.2 Domain Model for Household Activities: Target Predicates*

In this chapter we are interested in the acquisition and assessment of instances for the following event level predicates:

a) causal relationship: $Cause(e_1, e_2)$, $Effect(e_1, e_2)$

b) manner relationship: $Manner(e_1, e_2)$

c) temporal relationships: $Before(e_1, e_2)$, $After(e_1, e_2)$, $During(e_1, e_2)$

d) preference relationship: $isPreferableTo(e_1, e_2)$

e) general context relationship: $Context(e_1, e_2)$

f) no relationship: $NoRel(e_1, e_2)$.

As we see in the evaluation section (Section 6.5), these relationships - which are compiled based on the existent natural language processing literature - can be used to express essential n-ary predicates in useful, volunteer-created knowledge bases.

### *6.2.3 Obtaining the Background Knowledge*

The background knowledge is currently obtained in a semi-automatic manner, as described below. It is important to point out that the background knowledge can be obtained in a purely automatic manner (as detailed in the following subsection) - for the current project, we take advantage of the availability of a large amount of background knowledge in the

Table 6.1: **Basic Binary Event Relations and Corresponding Examples.** Notation: $e_1$, $e_2$ = state or action events.

| | |
|---|---|
| $Cause(e_1, e_2)$ | Cause([Person, Hear, Knock], [Person, Open, Door]) |
| $Effect(e_1, e_2)$ | Effect([Person, Open, Door], [Person, Hear, Knock]) |
| $Before(e_1, e_2)$ | Before[(Person, Leave, House), (Person, Open, Umbrella)] |
| $During(e_1, e_2)$ | During[(Person, Use, Water), (Person, Wash,Face)] |
| $After(e_1, e_2)$ | After[(Person, Open, Umbrella), (Person, Leave, House)] |
| $Context(e_1, e_2)$ | Context([Person, Brush, Teeth], [Person, Use, Toothpaste]) |
| $IsPreferableTo(e_1, e_2)$ | IsPreferableTo([Tea, Warm], [Tea, Cold]) |
| $NoRel(e_1, e_2)$ | No-rel([Person,Make,Tea], [Person,Water,Plant]) |

form of the Open Mind Indoor Common Sense database (described below) and focus on augmenting this knowledge as needed.

*The Open Mind Indoor Common Sense Knowledge Base: OMICS [44]*

The SRCS state estimation system that provides the evaluation framework for our commonsense fact acquisition and assessment procedures (see Section 6.5) started by using the basic facts available in the Open Mind Indoor Common Sense (OMICS) [44]. Similar to CyC [59], OMICS is a user-contributed database; unlike Cyc, which had a small dedicate team of humans adding information (and recently undertook efforts to acquire some facts from the Web [68]), OMICS allows users from all over the Internet to add facts. Users are presented with fill-in-the-blank questions such as "You *blank* when you are *blank*", with the expectation that the users will fill in, *e.g.*, "eat" and "hungry" in the two blanks. The sentence templates map into relations, *e.g.*, the `people (Action, Context)` relation, which may contain the `people(eat, hungry)`. A cleanup step is performed to remove noisy entries; the step uses WordNet-based distance functions to eliminate entries with entered values that are completely or mostly unrelated. The cleanup step also eliminates offensive entries.

The resulting data still contains some noise as well as irrelevant information; additionally, the collected information suffers from coverage gaps - in the following, we describe how the background knowledge used in this chapter is obtained by further cleaning up and restricting OMICS to a smaller, high-precision subset which is then augmented directly using the Web.

*Concepts* We start by compiling a set of *concepts* of interest. As seen in previous work, such a set can be compiled by retrieving *concrete* frequent terms from Web pages describing how-to information about household activities [87, 124]. In this chapter, we start with the basic object set provided by the volunteer created OMICS knowledge base and automatically eliminate the erroneous entries by using Web statistics to check that they are subclasses of the Object class (using the subclass assessment mechanism described in Chapter 3).

*Relations* Given the set of concepts of interest, a number of relation extraction methods can be used to identify corresponding relations involving the given concepts. In this chapter, we avail ourselves of the relation set provided by OMICS.

*Concept Properties* Given a set of concepts, the methods described in Chapter 5 can be employed to derive a set of explicit concept properties by parsing Web pages with how-to information, identifying potential properties and assessing them using Web statistics. In this chapter, we take advantage of the fact that the `state` facts in OMICS correspond to a large set of (object, property value) pairs (*e.g.*, `state(Door, Broken)`, `state(Door, Fixed)`, `state(Door, Open)`, `state(Door, Closed)`). Given the set of (object, property value) pairs provided by OMICS, our system uses the unsupervised adjective clustering mechanism described in Chapter 5 in order to uncover the implicit object properties referred to by the `state` information. In the examples above, the system would uncover the implicit properties `beFixed` (whose values are `Broken` and `Fixed` ) as well as `beOpen` (whose values are `Open` and `Closed`).

## 6.3 Assessing Instances of Commonsense Relations

We start by describing how we evaluate instances of *binary* event relations and continue by describing how we evaluate instances of *ternary* relations.

### 6.3.1 Binary Relation Assessment: Using Lexico-Syntactic Patterns

Let $R$ be a *binary* event-level relation and let $e_0 = [s_0, v_0, o_o]$ and $e_1 = [s_1, v_1, o_1]$ correspond to two *event descriptions* for the events of interest $e_0$ and $e_1$. As previously mentioned, events can contain more than one object $o$ - for simplicity, we focus on this simplest case in our write-up.

In the following we describe how we compute the probability that $R(e_0, e_1)$ holds.

Intuitively, $R(e_0, e_1)$ is more likely to hold if we can find instantiations of relation-specific lexico-syntactic patterns with complete or partial event descriptions for $e_0$ and $e_1$. As in previous chapters, we use the Web as a corpus and use the Google search engine to find hitcounts for queries based on such instantiated patterns. In the following, we describe this process in more detail.

Table 6.2: **Relation-specific Patterns.** Notation: $e_1$, $e_2$, $e_3$ = state or action events, [X], [Y] = argument slots.

| | |
|---|---|
| $Cause(e_1, e_2)$ | [Y] because [X], [X] leads to [Y], [Y] due to [X] |
| $Before(e_1, e_2)$ | [X] before [Y], [Y] after [X], [X] in order to [Y], [X] on the way to [Y] |
| $Manner(e_1, e_2)$ | [X] is a way to [Y], [Y] by [X], [X] as a means to [Y], [Y] using [X] |
| $Context(e_1, e_2)$ | [Y] while [X], [X] and [Y], [X] or [Y], [Y] if [X] |
| $IsPreferableTo(e_1, e_2)$ | [X] rather than [Y], [X] instead of [Y], [X] better than [Y] |

Let $partial_1$, $partial_2$ and $complete$ represent possible types of event descriptions: $partial_1$ and $partial_2$ refer to size 1 and 2 subsets of the complete event description while $complete$ refers to the complete description. For example, given $e_0 = [s_0, v_0, o_o]$, $[s_0, v_0]$ is a size 2 subset of $e_0$'s complete description and therefore is of type $partial_2$.

Let $D = \{partial_1, partial_2, complete\}$ represent the set of types of event descriptions. Let $PT$ represent the set of relation-specific patterns (see Table 6.2). Due to the high query cost of estimating the quantities of interest from the Web, we chose a small number of patterns with high precision rather than a larger number of patterns with lower average precision. The patterns were selected based on a combination of our previous research experience with some of these relations ([67], [91]) and bootstrapping for relations such as $Context()$.

We estimate $p(R(e_0, e_1))$ as follows:

$$p(R(e_0, e_1)) = \sum_{d \in D} w_d * p(R(e_0, e_1)|F_d),$$

where $d$ is a type of event description, $w_d$ reflects the confidence in $d$ (the sum of the weights is 1) and $F_d$ is a boolean feature computed as follows:

$F_d = 1$ iff $f_d(e_0, e_1) > th = threshold$ where $f_d(e_0, e_1)$ is defined as follows:

$$f_d(e_0, e_1) = max_{pt}(f_{d,pt}(e_0, e_1) * w_{pt}),$$

where $pt \in PT$, $f_{d,pt}(e_0, e_1)$ captures the information provided by the individual pattern $pt$ and $w_{pt}$ is the weight reflecting the importance of this individual pattern. We estimate the value of $f_{d,pt}(e_0, e_1)$ as follows:

$$f_{d,pt}(e_0, e_1) = \frac{\sum_{(s_i,s_j), s_i \in e_0, s_j \in e_1} Score(s_i, s_j, pt) * p(e_0|s_i, R) * p(e_1|s_j, R)}{\#(s_i, s_j)},$$

where $(s_i, s_j)$ is a pair of $d$-type subsets of $e_0$ and, respectively $e_1$, $Score(s_i, s_j, pt)$ is a quantity reflecting how likely $s_i$ and $s_j$ are to be related by means of pattern $pt$ (see below) and finally, the product of $p(e_0|s_i, R)$ and $p(e_1|s_j, R)$ is a measure of how likely $(s_i, s_j)$ is to be a good substitute for $(e_0, e_1)$ (if $s_i, s_j$ are very frequent and not highly correlated with a particular event pair, this measure will penalize their contribution, $Score(s_i, s_j, pt)$, accordingly).

$Score(s_i, s_j, pt)$ is computed as follows:

$$Score(s_i, s_j, pt) = \frac{hits(pt(s_i, s_j))}{hits(cocc(s_i, s_j))}.$$

$cooc(s_i, s_j)$ denotes event co-occurrences in Web text: in order to find such co-occurrences, we employ the queries "$s_i * s_j$ " and "$s_j * s_i$ " (or, if the returned hitcounts are 0, " $s_i + s_j$ " and " $s_j + s_i$ ").

$p(e|s, R)$ is computed as seen below:

$$p(e|s, R) = \frac{\sum_{pt' \in PT} hits(pt'(e)}{\sum_{pt'' \in PT} hits(pt''(s))}$$

Initial experiments have shown that estimating these conditional probability terms can be problematic due to the sparse data problem (even when computing these terms at Web scale). In order to mitigate this problem, we ignore the $R$ term and compute $p(e|s)$ instead whenever necessary.

*Thresholds and weights* The thresholds used to convert $f_d(e_0, e_1)$ quantities into boolean features are estimated based on a set of 20 positive and 20 negative examples for the relation of interest. In our experiments, the $w_d$ weights - reflecting the importance of type $d$ event descriptions - were manually set (we required that $w_{partial_1} > w_{partial_2} > w_{complete}$), but we are in the process of experimenting with values learned on the training set mentioned above. Finally, each of the high-precision relation-specific pattern was given the same $w_{pt}$ weight (the default value was 0.9).

### *6.3.2 Binary Relation Assessment: Using Relational Definitions*

In the previous subsection, we used lexico-syntactic patterns to directly check whether a particular event-level relation holds. While lexico-syntactic patterns are useful for more specific relations such as *Cause*, higher-level predicates such as $isPreferableTo()$ could benefit from additional sources of evidence.

One potential such evidence source is represented by *relational definitions*, taking the form of Horn formulas. In our case, the predicates in the formulas are defined on top of the existent domain model (which can be augmented as described below). The formulas themselves can be supplied manually or can be learned using traditional ILP techniques or recent methods such as Markov Logic Network structure learning; the results in this chapter use a small number of manually supplied definitions describing the relation of interest.

*Example*

Here is an example of a relational definition for a higher-level event relation:

$$\begin{aligned} isPreferableTo(e_0, e_1) \quad := \quad & isState(e_0) \wedge isState(e_1) \wedge sameImplicitProperty(e_0, e_1) \wedge \\ & positiveSemanticOrientation(e_0) \wedge \\ & negativeSemanticOrientation(e_1) \end{aligned}$$

In our case, each literal will have an associated probability (some of them will have probability 1.0 - for example, an event is or is not a state event - while others will have probabilities less than 1.0 - for example, the semantic orientation predicates). The probability associated with the value of $isPreferableTo(e_0, e_1)$ can be computed using any of a number of combination functions (mean rule, product rule, etc.).

Some relations have multiple definitions, as seen in the example below - additionally, definitions can have associated parameters indicating how likely they are to capture the desired semantics of the relation.

$$\begin{aligned} context(e_0, e_1) \quad := \quad & cause(e_0, e_1)|effect(e_0, e_1)|before(e_0, e_1) \\ & after(e_0, e_1)|during(e_0, e_1)|manner(e_0, e_1) \end{aligned}$$

If a particular relation has $k$ alternative definitions, the probability of a relation instance can be estimated in a number of ways - for simplicity, we use the Noisy-Or and Noisy-Max functions.

Let $p(R(e_0, e_1), D_R)$ correspond to the probability of a particular relation instance computed using evidence from available relational definitions. Let $p(R(e_0, e_1), P_R)$ correspond to the probability of a relation instance computed using lexico-syntactic information, as detailed in the previous sections.

We combine the probabilities corresponding to these sources of evidence in order to derive the final value of a probability associated with a particular instance:

$$p(R(e_0, e_1)) = \alpha_0 * p(R(e_0, e_1), P_R) + \alpha_1 * p(R(e_0, e_1), D_R),$$

where $\alpha_0$ and $\alpha_1$ are parameters indicating how much importance to attach to a particular source of evidence (if no evidence from a particular source is available, the corresponding parameter is 0 and the remaining parameter is set to 1).

### *6.3.3 Ternary Relation Assessment: Using Relational Definitions*

Most of the commonsense relations of interest to us are binary - the exception is represented by the *stateChange* relation below, which is an important relation in the Open Mind commonsense database. Additionally, our current work explores the learning of other utility predicates: ternary and n-ary relations that correspond to constraints involving more than 2 events: for example, *activitySteps($e_0$, $e_1$, $e_2$)* denotes the fact that $e_0$, $e_1$ and $e_2$ are consecutive steps of a particular human activity.

An example relational definition used for the *stateChange* relation can be seen below:

$$\begin{aligned} stateChange(e_0, e_1, e_2) \quad := \quad & isState(e_0) \wedge isAction(e_1) \wedge isState(e_2) \wedge \\ & cause(e_0, e_1) \wedge cause(e_1, e_2) \wedge \\ & sameImplicitProperty(e_0, e_2) \end{aligned}$$

For the purposes of this chapter's experimental evaluation, instances of ternary relations were assessed using the corresponding set of relational definitions as in the case of previously described binary relations.

### *6.3.4 Labeling Potential Instances with Appropriate Relation Names*

Section 6.4 describes our procedure for iterative mining of complex relation instances - as part of this procedure, the system labels each potential relation instance with the appropriate relation name (*e.g.*, Cause ) as described below.

Given the set of target predicates $TR$ and an event pair $(e, e')$, the system assigns the event pair the relation label $R \in TR$ such that

$R = argmax_R[p(R(e, e'))]$, where $p(R(e, e'))$ is computed as described above.

In the experiments described in this chapter, we used the simple method described above in order to label potential relation instances; however, a more accurate labeling procedure would take into account the dependencies among the target relations, as described in our on-going work section (Section 6.7).

We have described how potential instances of event-level relations are assessed using Web-scale statistics - in the following, we describe our iterative instance acquisition mechanism.

## *6.4 Mining Commonsense Facts from the Web*

In this section we describe how commonsense facts can be automatically acquired from the Web in the context of existent *background knowledge* (in the form of a populated *basic domain model*).

### *6.4.1 Finding Commonsense Facts: An Overview*

Section 6.2 describes the basic domain model as consisting of

a) $O$, a set of *objects* and *object properties* together with *property values*,

b) $E$, a set of *action* and *state events*,

c) $BR$, a set of background event-level predicates such as *positiveSemanticOrientation()*.

Given this *basic* domain model, the system augments it by finding pairs of known events that correspond to instances of *event-level commonsense relations* (see Table 6.1).

Figure 6.1 contains an overview of this instance acquisition process.

Given the basic domain model $< O, E, BR >$ and the set of target relations $TR$, the space of potential instances to be labeled is quite large. In order to prune it, the system takes as input a set of queries of interest $Q$ and restricts the set of objects $O$ to contain *observable* objects (objects whose use is directly tracked using sensors).

The system uses the iterative procedure described in Figure 6.1 to find instances of target relations. At each iteration, the events matching the target queries (*e.g.*, *[person, swallow, pill]* matches the query *swallow*) are selected: for each such event $e$, all events $e'$ also matching target queries or, alternatively, containing observable objects, are also selected.

The system examines each resulting event pair $(e, e')$ in order to decide whether it's a potential instance of a target relation - at this step, a cheap classifier that uses mostly background knowledge to decide whether $e$ and $e'$ are conceptually related is employed. If $e$ and $e'$ are found to be related, a second Web-based classifier is used to decide if $(e, e')$ is an instance of one of the target relations. If so, $e'$ is added to the set of target queries $Q$.

Once the set of events $e$ has been exhausted, the process is repeated with the updated target query set as many times as desired.

In the following, we describe how potential instances of target relations are identified.

**FocusedFactMining(Q, O, E, BR, TR, numIterations)**

```
1  $i = 0$; $F = \{\}$;
2   while $i < numIterations$
3      $\forall e \in E$ s.t. $matchesQuery(e, Q)$
4        $\forall e' \in E$ s.t. $matchesQuery(e', Q)$ or $containsObject(e', O)$
5          if $potentialTargetRelationInstance(e, e', R)$
6            $F \leftarrow F \cup generateRelationInstances(e, e', R)$;
7            $Q \leftarrow Q \cup \{e'\}$;
8      $i++$;
9  return $F$;
```

Figure 6.1: **Overview of the Focused Fact Mining Process.** Notation: Q = set of queries of interest, O = set of objects of interest, BR = set of background relations, TR = set of target relations, E = set of events, e, e' = events in E, F = set of facts (final instances of event level relations), numIterations = number of iterations.

.

### *6.4.2 Identifying Potential Facts*

Given a pair of events $(e, e')$, our system first uses a simple decision tree classifier $C_{rel}$ to determine whether $(e, e')$ are likely to be linked by means of a direct relation. If so, the appropriate relation label is computed as described in Section 6.3.

The $C_{rel}$ classifier checks whether two events $e$ and $e'$ are directly related as follows:

a) If $overlap(e, e') \geq ct$, $e$ and $e'$ are considered to be related - $overlap(e, e')$ is defined as follows:

$$overlap(e, e') = max(\sum_{t \in e} p(contains(e', t))/size(e), \sum_{t' \in e'} p(contains(e, t'))/size(e'))$$

where $size(e)$ denotes the number of terms in $e$ and $p(contains(e, t'))$ is the probability that $e'$ contains term $t$ from $e$. This probability is 1.0 if $t'$ appears in $e$ - if $e$ contains a term $t$ such that $t$ and $t'$ are linked by means of a background relation $R$, the probability is set to that of the corresponding instance of $R$.

b) If $overlap(e, e') < ct$ but $PMIScore(e, e') \geq ct'$, $e$ and $e'$ are considered to be related, where:

$$PMIScore(e, e') = \frac{hits(cooc(e, e'))}{hits(e) * hits(e')}$$

.

$cooc(e, e')$ denotes event co-occurrences in Web text: in order to find such co-occurrences, we employ the queries “e * e’ “ and “e’ * e “ (or, if the returned hitcounts are 0, “ e + e’ “ and “ e’ + e ”).

c) If $overlap(e, e') < ct$ and $PMIScore(e, e') < ct'$, $e$ and $e'$ are unrelated.

The identified potential facts are labeled with the appropriate relation label as described in Section 6.3.

## 6.5 Results

This section describes our experimental results - we show that:

a) Web statistics can be used to compute useful scores for available commonsense information provided by volunteers - when using this *assessed* version of the data, the recently introduced state estimation system SRCS (see subsection 6.5.1) sees a 13% increase in precision over using the initial version of the data.

b) High precision instances of useful commonsense relations can be automatically acquired from the Web - the performance of the SRCS system when using automatically mined data is similar to the performance of SRCS when using the assessed version of the volunteer-supplied data.

In the following we describe our evaluation and results in more detail.

### 6.5.1 Using Commonsense Knowledge For State Recognition with SRCS

Our evaluation uses the recently introduced SRCS system for state estimation [86]. SRCS operates by collecting sensory input from a user and employing statistical inference methods to reason about various predefined facts about the state of the world (*e.g.*, “Is the light on?”, “Is the user in the kitchen ?”, “Is the user hungry ?”). The model translating between observations and abstract states is acquired from existing hand-created commonsense databases. As mentioned earlier in this chapter, SRCS relies on the information provided by the OMICS knowledge base. Currently, SRCS makes use of a subset of the OMICS relations (see Table 6.3) although more are being integrated in the system's formalism. As seen in Table 6.3, the OMICS relations of interest can be expressed in terms of the basic target predicates defined in section 6.2. In order to assess potential instances of OMICS predicates, we first check the typing constraints: if they are not met, the instance is not a correct instance - if they are met, we assess the corresponding target predicate as described in the previous sections.

Table 6.3: **OMICS Relations.** Notation: $e_1$, $e_2$, $e_3$ = state or action events.

| **OMICS Predicate** | Type Constraints | **Basic Predicate** |
|---|---|---|
| $Cause_{OMICS}(e_1, e_2)$ | $Action(e_1), State(e_2)$ | $Cause(e_1, e_2)$ |
| $Response_{OMICS}(e_1, e_2)$ | $State(e_1), Action(e_2)$ | $Cause(e_1, e_2)$ |
| $People_{OMICS}(e_1, e_2)$ | $State(e_1), Action(e_2), agentTypePerson(e_1)$ | $Cause(e_1, e_2)$ |
| $Context_{OMICS}(e_1, e_2)$ | $N/A$ | $Context(e_1, e_2)$ |
| $Desire_{OMICS}(e_1, e_2)$ | $State(e_1), State(e_2)$ | $isPreferableTo(e_1, e_2)$ |

In the following, we describe two different evaluations that show the value of assessing and mining commonsense information using Web data.

### 6.5.2 *Using Web Statistics to Assess Manually Supplied Commonsense Facts*

Table 6.4: **The set of activities for which experimental trace data was collected.**

| | | |
|---|---|---|
| brush teeth | take medication | water plants |
| shave your face | take a shower | watch television |
| dust shelves | eat cereal | groom hair |
| write a letter | make cereal | wash windows |
| | make tea | |

The OMICS knowledge base contains a lot of valuable information, but since the data is supplied by volunteers it contains some noisy or not very relevant information. We use Web statistics to assess the manually created instances of commonsense relations, which are then supplied to the SRCS system - as seen in Table 6.5, the use of the assessed instances results in a 13% precision improvement.

For the experimental evaluation whose results are summarized in Table 6.5, traces of the iBracelet's output were collected in an experimental setting as worn by three users while performing various daily activities in a simulated home environment. The list of activities performed can be seen in Table 6.4. A total of 5-7 minutes worth of performance of each activity was collected, for a total of approximately 70-75 minutes of data. These traces were divided into time slices of 2.5 seconds - reasoning was to be performed over each of the time slices. For these activities, we selected a set of 24 Boolean variables from the OMICS database which represented them and recorded their "truth" value as being true of false for each interval of time in the trace - this human labeling of these traces is the source of ground truth.

Table 6.5: **The Impact of Assessed Commonsense Knowledge on the Performance of** SRCS **: Assessed Information Significantly Improves the Performance of the State Estimation System.** Notation: Precision, Recall and Accuracy are the performance measures of interest; SRCS/original data = SRCS using the original set of OMICS facts ; SRCS/assessed data = SRCS using the assessed version of the OMICS facts; Random = randomly labeling the variables in the test set as True or False.

| **Method** | **Accuracy** | **Recall** | **Precision** |
|---|---|---|---|
| Random | 50% | 50% | 8% |
| SRCS/original data | 80% | 46% | 18% |
| SRCS/assessed data | 88% | 53% | 31% |

We compared the human labeling of these traces for the 24 variables of interest to the labeling automatically obtained from SRCS when using a) the original set of OMICS facts and then b) the assessed set of OMICS facts (that is, the set of OMICS facts together with the probabilities computed using Web statistics).

The measures we consider are the standard information retrieval measures of precision and recall with respect to discovery of True labels in addition to the labeling accuracy measure; in the case of the state recognition problem, most of the variables are false most of the time ( 94% of labels are false) and finding true variables is of greater interest.

Assessing OMICS facts considerably improved the accuracy and precision of the system - much of the difference in results is caused by a decrease in positive labels (that is, variables labeled "true"); while this lowers the recall slightly, the precision is considerably improved by the decrease in false positives.

Table 6.6: **Identifying Highly Relevant Commonsense Information**. Notation: **Rel** = relation name; **Num. Facts** = number of OMICS facts for the relation of interest; % **Facts** ($p > 0.7$) = percentage of original facts whose assigned probability is greater than or equal to 0.7; **Prec. Orig. Facts** = precision of original facts; **Prec. Facts** ($p \geq 0.7$) = precision of facts with prob. greater or equal to 0.7.

| **Rel** | **Num. Facts** | % **Facts** ($p \geq 0.7$) | **Prec. Orig. Facts** | **Prec. Facts** ($p \geq 0.7$) |
|---|---|---|---|---|
| $cause_{OMICS}$ | 6749 | 70% | 84% | 97% |
| $context_{OMICS}$ | 3014 | 64% | 96% | 100% |
| $response_{OMICS}$ | 6903 | 56% | 96% | 100% |
| $people_{OMICS}$ | 1235 | 68% | 94% | 97% |
| $desire_{OMICS}$ | 1186 | 33% | 90% | 96% |
| *Average* | 19087 | 61% | 92% | 98% |

As seen in Table 6.5, the mined probabilities are effective in weeding out correlations of lower quality in the OMICS data. Even outside the SRCS system, the advantages of assessing OMICS information can be quantified as seen in Table 6.6: given a set of 250 OMICS facts (50 per OMICS relation), this table shows that assessed OMICS facts with high associated probabilities are more relevant than unassessed OMICS facts. The precision measure not only takes into account the truth value of a fact, but also reflects the perceived importance of a fact (for example, a toothbrush being opened is an unlikely *cause* of a toothbrush being used to brush teeth).

Table 6.7 contains additional examples of incorrect or unlikely instances of a commonsense relation. Typical errors include typing or spelling errors ("launcher" vs. "lawn chair") and mistaking the cause for the effect ("people watch television because people know latest news", instead of viceversa). Additionally, people input sometimes irrelevant (or potentially wrong information) (the door being unattached is preferable to the door being broken).

Meanwhile, the OMICS facts with highest assigned probabilities tend to represent useful dependencies among states of the same object (" if the coffee maker is plugged in, the coffee maker is on") or dependencies among closely related objects ("if the cushions are soft, the couch is soft").

Table 6.7: **Examples of Incorrect or Less than Relevant Information Eliminated by Using Web-based Statistics.**

| **Relation** | **Example** |
|---|---|
| $cause_{OMICS}$ | (toothbrush,opened) → (toothbrush, used to brush, teeth)<br>(phone,ringing) → (phone,connected)<br>(nightstand,used to keep lamp) → (lamp, usable for reading) |
| $context_{OMICS}$ | (broken,launcher) → (replace,lawn chair) |
| $people_{OMICS}$ | (people,know,latest news) → (people,watch,television) |
| $desire_{OMICS}$ | (door,broken) → (door,unattached) |

### *6.5.3 Using the Web to Replace and Augment Existent Commonsense Knowledge Bases*

A second evaluation looked at whether high quality instances of event-level relations can be acquired from the Web: we measured the quality of the acquired instances both directly and by supplying them to the SRCS system, which then used them as its background knowledge in order to solve its state recognition task.

As we can see in Table 6.8, directly measuring the precision of the acquired instances shows that the found facts are of good quality - additionally, we found many facts missing from OMICS, which shows that the Web is a valuable resource for automatically augmenting existent knowledge bases. As previously mentioned, the preliminary experiments on which we report here have used the data obtained after only 2 iterations of our focused fact mining routine: running the fact mining routine for more iterations will of course result in an increased yield.

Additionally, we show that the acquired relation instances are valuable in the context of the state estimation task solved by SRCS (see Table 6.9 for a summary of the corresponding results). In order to do so, we repeated the experiments described in the previous sections, with a few modifications: a) we used both the original version of the SRCS system and an improved version which includes an additional learning component, b) we ran SRCS with the assessed version of the original OMICS data, the mined commonsense data and the two data sets combined.

As seen in Table 6.9, using the mined data and using the assessed version of the original OMICS data lead to similar performance numbers. This is an encouraging result, as it shows that commonsense relation models mined from the Web can replace manually supplied

Table 6.8: **Mining Commonsense Facts**. Notation: Precision = precision of extracted facts,Yield = number of extracted facts, Proportion of Novel Facts =% of mined facts that are not in OMICS. The results are reported for facts with $p > 0.75$ found after 2 iterations of the fact mining routine.

| **Relation** | **Precision** | **Yield** | **Proportion of Novel Facts** |
|---|---|---|---|
| $cause_{OMICS}$ | 76% | 424 | 68% |
| $response_{OMICS}$ | 73% | 253 | 74% |
| $context_{OMICS}$ | 71% | 326 | 71% |
| $desire_{OMICS}$ | 82% | 400 | 66% |
| $people_{OMICS}$ | 82% | 153 | 94% |
| $stateChange_{OMICS}$ | 85% | 459 | 77% |

Table 6.9: **The Impact of Mined Commonsense Data on the SRCS system: Using the mined event data in addition to the assessed original data improves the system's performance.** Notation: Orig = SRCS using the assessed original data, Mined = SRCS using the mined event data, Orig/Mined = SRCS using both the assessed original data and the mined event data, Learning = boolean variable reflecting the system's use of its learning component. We report on the Precision, Recall and Accuracy measures.

| **Method** | **Learning** | **Accuracy** | **Precision** | **Recall** |
|---|---|---|---|---|
| Orig | No | 82% | 25% | 51 % |
| Orig | Yes | 83% | 24% | 67 % |
| Mined | No | 84% | 23% | 52 % |
| Orig/Mined | No | 86% | 30% | 83 % |

relation models for this task.

Using the two data sets together results in significant improvements in recall and precision over the best results of the SRCS system when using only one or the other of the two sets. This result reflects the fact that our system was able to augment the manually supplied set of commonsense relation instances with additional high quality information from the Web.

## *6.6 Related Work*

*The Web as a corpus.* Recent work on Cyc [68] and activity recognition [124] has shown that it is possible to extract common sense information from the Web. In the case of the former work, Cyc follows the lead of KnowItAll in using the Web to extract instances of simple binary relations (simple factoids) from the Web - we focus on more complex, event-level relations suitable for activity and state recognition applications. The latter work is restricted to capturing the very simple and vague *Use* binary relationship between activities and objects - in this chapter, we extract more complex and varied information suitable for state recognition.

*Semantic Relationship and Inference Rules Extraction.* One of the important commonsense relations we discussed in this chapter is the *Cause* relation. While NLP researchers (including the author of this thesis) have started addressing the extraction of causal relationships ([42], [51], [66], [67]), there are still many advances to be made in this area. So far, people have focused on extracting instances of causal relations at the level of phenomena described by simple nouns (*e.g.*, "flood" potentially causes "hunger") or, alternatively, the more vague type of discourse causal relations (sentence A is connected to sentence B my means of the Cause discourse relation rather than by means of the Contrast relation). In this chapter we investigate an additional type of Causal relation: concrete, event-level Causal relations in the context of background knowledge. In fact, the absence of background knowledge has greatly limited the types of causal information people were able to extract from text in previous work - most efforts focus on open domain, general causal links (such as the *flood causes hunger* example) or sentence-level links (sentence A causes sentence B).

Most previous projects focus on mining *explicit* causal relations - relations explicitly stated at the level of a sentence or relations between two consecutive sentences expressed and expressed by connective markers (*e.g.*, *because*, *because of*). We are interested in both explicit and *implicit* causal relations (instances of causal relations that can only be assessed at the level of the Web rather than found in one particular sentence). Additionally, we use a different mechanism for assessing the extracted data and we make use of automatically acquired IS-A and Part-of information.

A related effort is the extraction of inference rules from text ([82], [112]): [82] is limited to paraphrases while [112] uses a statistical approach to validate potential If-Then rules based on word co-occurence information extracted from Japanese text. While we also take advantage of corpus statistics when assessing potential relation instances, we validate more complex instances and do so by combining statistical and relational information - our work can be seen as extending that in [112] and combining it with insights from [72], a paper that addresses the assessment of n-ary relation instances that are decomposable into sets of binary relation instances.

## ***6.7 Work in Progress***

There are many directions for future work and we are currently pursuing a few, as detailed in the following.

We are looking into the automatic evaluation of the context-specific semantic orientation of a particular event $e$ *e.g.*, $contextSpecificPositiveSO(e, e_c)$, where $e_c$ represents the context of interest. We are also looking at the assessment of additional commonsense predicates in the OMICS database, as they are being integrated into the SRCS formalism.

In order to improve the quality of the mined models, we are currently testing a scheme that combines a *linear label propagation* method with active learning heuristics in order to both improve the precision and yield of our mining routine as well as reduce the number of necessary search engine queries (*under preparation*).

# BIBLIOGRAPHY

[1] Calo: Cognitive assistant that learns and organizes. Technical report, SRI International, 2005.

[2] E. Agichtein and L. Gravano. Snowball: Extracting Relations from Large Plain-Text Collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, pages 85–94, San Antonio, Texas, 2000.

[3] E. Agirre, O. Ansa, D. Martinez, and E. Hovy. Enriching WordNet concepts with topic signatures. In *Proceedings of the NAACL worshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations*, 2001.

[4] A. Almuhareb and M. Poesio. Attribute-based and value-based clustering: An evaluation. In *Procs. of EMNLP*, pages 158–165, 2004.

[5] N. Ashish and C. Knoblock. Semi-automatic wrapper generation for Internet information sources. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*, pages 160–169, Los Alamitos, CA, June 1997. IEEE-CS Press.

[6] T. Baldwin and F. Bond. Learning the countability of english nouns from corpus data. In *Proceedings of ACL*, 2003.

[7] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the Web. In *Proceedings of the IJCAI*, 2007.

[8] M. Bauer, D. Dengler, and G. Paul. Instructible information agents for web mining. In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, January 2000.

[9] M. Berland and E. Charniak. Finding parts in very large corpora. In *Procs. of ACL*, pages 57–64, 1999.

[10] C. Blake and W. Pratt. Collaborative information synthesis. In *Proceedings of ASIST*, 2002.

[11] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, Madison, Wisconsin, 1998.

[12] E. Brill. Some Advances in Rule-Based Part of Speech Tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 722–727, Seattle, Washington, 1994.

[13] S. Brin. Extracting Patterns and Relations from the World Wide Web. In *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, pages 172–183, Valencia, Spain, 1998.

[14] R. Bunescu and R. Mooney. Collective information extraction with Relational Markov Networks. In *Proceedings of ACL*, pages 439–446, 2004.

[15] M.J. Cafarella, M. Banko, and O. Etzioni. Relational web search. Unpublished, UW CSE, April 2006.

[16] M.E. Califf and R.J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[17] Jinxiu Chen, Donghong Ji, Chew L. Tan, and Zhengyu Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 129–136, 2006.

[18] T. Chklovski and P. Pantel. VerbOcean: Mining the Web for Fine-Grained Semantic Verb Relations. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pages 33–40, 2004.

[19] K.W. Churck and P. Hanks. Word association norms, mutual information and lexicography. In *Proceedings of the 27th Annual Conference of the Association of Computational Linguistics*, pages 76–83, 1989.

[20] K.W. Churck, P. Hanks, and D. Hindle. Using statistics in lexical analysis. In *Lexical Acquisition: Exploiting On-Line Resources to Build a Lexicon*, pages 115–164. Lawrence Erlbaum, 1991.

[21] F. Ciravegna. Adaptive Information Extraction from Text by Rule Induction and Generalisation. In *Procs. of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 1251–1256, Seattle, Washington, 2001.

[22] F. Ciravegna, A. Dingli, D. Guthrie, and Y. Wilks. Integrating Information to Bootstrap Information Extraction from Web Sites. In *Procs. of the IIWeb Workshop at the 19th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 9–14, Acapulco, Mexico, 2003.

[23] M.Musen C.J. Hou, N.Noy. A template-based approach towards acquisition of logical sentences. *Intelligent Information Processing*, pages 77–89, 2002.

[24] M. Collins and Y. Singer. Unsupervised Models for Named Entity Classification. In *Procs. of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–111, Maryland, USA, 1999.

[25] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence 118(1-2)*, pages 69–113, 2000.

[26] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Sean Slattery. Learning to extract knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[27] A. Culotta, A. McCallum, and J. Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the HLT-NAACL*, 2006.

[28] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. Tomlin, and J. Zien. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *Proceedings of the 12th International Conference on World Wide Web*, pages 178–186, Budapest, Hungary, 2003.

[29] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the Eleventh International WWW Conference*, 2002.

[30] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[31] D. Downey, S. Soderland, and O. Etzioni. A probabilistic model of redundancy in information extraction. In *IJCAI*, 2005.

[32] M. Erdmann, A. Maedche, H. Schnurr, and S. Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *Proceedings of the COLING Workshop on Semantic Annotation and Intelligent Content*, 2000.

[33] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in knowitall (preliminary results). In *Proceedings of the World Wide Web Conference*, 2004.

[34] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web - an experimental study. *Artificial Intelligence Journal*, 2005.

[35] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Methods for domain-independent information extraction from the Web: An experimental comparison. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-2004)*, pages 391–398, 2004.

[36] H. Fargier and J. Lang. A constraint satisfaction framework for decision under uncertainty. In *Procs. of UAI*, pages 167–174, 1995.

[37] D. Faure and C. Nedellec. A corpus-based conceptual clustering method for verb frames and ontology acquisition. In *Proceedings of the LREC Workshop on Adapting Lexical and Corpus Resources to Sublanguages and Applications*, 1998.

[38] Jenny Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings o the ACL*, 2005.

[39] D. Freitag and A. McCallum. Information Extraction with HMMs and Shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, Orlando, Florida, 1999.

[40] M. Gamon. Sentiment classification on customer feedback data: Noisy data, large feature vectors and the role of linguistic analysis. In *Procs. of COLING*, pages 841–847, 2004.

[41] A. Gangemi, R. Navigli, and P. Velardi. The Ontowordnet Project: Extension and Axiomatization of Conceptual Relations in WordNet. In *International Conference on Ontologies, Databases and Applications of Semantics*, 2003.

[42] R. Girju. Automatic detection of causal relations for question answering. In *Proceedings of the ACL*, 2003.

[43] N. Guarino and C.A. Welty. An Overview of OntoClean. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 151–172. Springer, 2004.

[44] R. Gupta and M.J. Kochenderfer. Common Sense Data Acquisition for Indoor Mobile Robots. In *Proceedings of AAAI*, pages 605–610, 2004.

[45] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Proceedings of CIDR*, 2003.

[46] V. Hatzivassiloglou and K. McKeown. Towards the automatic identification of adjectival scales: clustering adjectives according to meaning. In *Procs. of ACL*, pages 182–192, 1993.

[47] V. Hatzivassiloglou and K. McKeown. Predicting the semantic orientation of adjectives. In *Procs. of ACL/EACL*, pages 174–181, 1997.

[48] M. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Procs. of the 14th International Conference on Computational Linguistics*, pages 539–545, Nantes, France, 1992.

[49] M. Hu and B. Liu. Mining and Summarizing Customer Reviews. In *Procs. of KDD*, pages 168–177, Seattle, WA, 2004.

[50] R.A. Hummel and S.W. Zucker. On the foundations of relaxation labeling processes. In *PAMI*, pages 267–287, 1983.

[51] T. Inui. Acquiring causal knowledge from text using connective markers. In *Ph.D Thesis*, 2004.

[52] R. Jones, R. Ghani, T. Mitchell, and E. Riloff. Active learning for information extraction with multiple view feature sets. In *Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining*, 2003.

[53] S. Kim and E. Hovy. Determining the sentiment of opinions. In *Procs. of COLING*, 2004.

[54] N. Kobayashi, K. Inui, K. Tateishi, and T. Fukushima. Collecting Evaluative Expressions for Opinion Extraction. In *Procs. of IJCNLP*, pages 596–605, 2004.

[55] B. Krulwich. The BargainFinder agent: Comparison price shopping on the Internet. In J. Williams, editor, *Bots and Other Internet Beasties*, chapter 13. SAMS.NET, 1996.

[56] D. Kushal, S. Lawrence, and D. Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Procs. of WWW*, 2003.

[57] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 729–737. San Francisco, CA: Morgan Kaufmann, 1997.

[58] C. T. Kwok, O. Etzioni, and D. Weld. Scaling Question Answering to the Web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.

[59] D. Lenat and R.V. Guha. *Building Large, Knowledge-based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1990.

[60] H. Li and N. Abe. Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics*, 24(2), 1998.

[61] D. Lin. Dependency-based evaluation of MINIPAR. In *Procs. of ICLRE'98 Workshop on Evaluation of Parsing Systems*, 1998.

[62] A. Maedche and S. Staab. Discovering conceptual relations from text. In *Proceedings of the 13th European Conference on Artificial Intelligence*, 2000.

[63] A. Maedche and S. Staab. Learning ontologies for the semantic web. In *Proceedings of the Second International Workshop on the Semantic Web*, 2001.

[64] B. Magnini, M. Negri, and H. Tanev. Is It the Right Answer? Exploiting Web Redundancy for Answer Validation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 425–432, 2002.

[65] Gideon Mann and David Yarowsky. Multi-field information extraction and cross-document fusion. In *Proceedings of the ACL*, 2005.

[66] D. Marcu and A. Echihabi. An unsupervised approach to recognizing discourse relations. In *ACL*, 2002.

[67] D. Marcu and A. Popescu. Towards developing probabilistic generative models for reasoning with natural language representations. In *CICLING*, 2005.

[68] C. Matuszek, M. Witbrock, R.C. Kahlert, J. Cabral, D. Schneider, P. Shah, and D. Lenat. Searching for common sense: Populating Cyc from the web. In *Proceedings of AAAI*, 2005.

[69] M.Banko, E.Brill, S.Dumais, and J.Lin. AskMSR: Question Answering Using the Worldwide Web. In *Proceedings of 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*, pages 7–9, Palo Alto, California, 2002.

[70] A. McCallum. Efficiently Inducing Features of Conditional Random Fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 403–410, Acapulco, Mexico, 2003.

[71] A. McCallum and B. Wellner. Object consolidation by graph partitioning with a conditionally trained distance metric. In *Proceedings of SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 19–24, 2003.

[72] Ryan McDonald, Fernando Pereira, Seth Kulick, Scott Winters, Yang Jin, and Pete White. Simple algorithms for complex relation extraction with applications to biomedical IE. In *Proceedings of the ACL*, 2005.

[73] G. Miller. WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1991.

[74] S. Morinaga, K. Yamanishi, K. Tateishi, and T. Fukushima. Mining product reputations on the web. In *Procs. of KDD*, pages 341–349, 2002.

[75] P. Nakov and M. Hearst. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *CoNLL*, 2005.

[76] P. Nakov and M. Hearst. A study of using search engine hits as a proxy for n-gram frequencies. In *RANLP*, 2005.

[77] P. Nakov and M. Hearst. Using the web as an implicit training set: Application to structural ambiguity resolution. In *HLT-NAACL*, 2005.

[78] K. Nigam, J. Lafferty, and A. McCallum. Using Maximum Entropy for Text Classification. In *Procs. of IJCAI-99 Workshop on Machine Learning for Information Filtering*, pages 61–67, Stockholm, Sweden, 1999.

[79] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Learning to classify text from labeled and unlabeled documents. In *Proceedings of the Fifteenth Conference of the American Association for Artificial Intelligence*, pages 792–799, 1998.

[80] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, pages 271–278, 2004.

[81] Lee L. Pang, B and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Procs. of EMNLP*, pages 79–86, 2002.

[82] Pantel.P and Lin.D. Discovery of inference rules from text. In *Proceedings of ACM-SIGKDD*, 2001.

[83] M. Pasca. Finding Instance Names and Alternative Glosses on the Web: WordNet Reloaded. In *Proceedings of CICLING*, 2005.

[84] M. Pennacchiotti and P. Pantel. A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. In *Proceedings of Inference in Computational Semantics*, pages 87–96, 2006.

[85] M. Pennacchiotti and P. Pantel. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *Proceedings of the ACL*, 2006.

[86] B. Pentney, A. Popescu, S. Wang, M. Philipose, and H. Kautz. Sensor-based Understanding of Daily Life via Large-Scale Use of Commonsense. In *Proceedings of AAAI*, 2006.

[87] M. Perkowitz, M. Philipose, D. Patterson, and K. Fishkin. Mining Models of Human Activities from the Web. In *Proceedings of WWW*, 2004.

[88] W. Phillips and E. Riloff. Exploiting strong syntactic heuristics and co-training to learn semantic lexicons. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

[89] A. Popescu and O. Etzioni. Large-Scale Ontology Extension from the Web. In *University of Washington - Microsoft Research Symposium*, 2004.

[90] A. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Proceedings of the EMNLP*, 2005.

[91] A. Popescu and O. Etzioni. In-depth review mining. In *Text Mining and Natural Language Processing*. Spring, 2006.

[92] A. Popescu, A. Yates, and O. Etzioni. Class extraction from the World Wide Web. In *AAAI-04 Workshop on Adaptive Text Extraction and Mining*, pages 68–73, 2004.

[93] W. Pratt and M. Yetisgen-Yildiz. LitLinker: Capturing connections across the biomedical literature. In *Proceedings of K-Cap*, 2003.

[94] A. Rangarajan. Self annealing and self annihilation: unifying deterministic annealing and relaxation labeling. In *Pattern Recognition, 33:635-649*, 2000.

[95] E. Riloff. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–6, 1993.

[96] E. Riloff and R. Jones. Learning Dictionaries for Information Extraction by Multi-level Bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 474–479, 1999.

[97] E. Riloff, J. Wiebe, and T. Wilson. Learning Subjective Nouns Using Extraction Pattern Bootstrapping. In *Procs. of CoNLL*, pages 25–32s, 2003.

[98] B. Roark and E. Charniak. Noun-phrase co-occurence statistics for semi-automatic semantic lexicon construction. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*, 1998.

[99] L. Schubert. Can we derive general world knowledge from texts. In *Procs. of Human Language Technology Conference*, 2002.

[100] Y. Shinyama and S. Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the HLT-NAACL*, 2006.

[101] A. Siddharthan. Resolving pronouns robustly: Plumbing the depths of shallowness. In *EACL 2003 Workshop on Computation Treatments of Anaphora*, 2003.

[102] Parag Singla and Pedro Domingos. Object identification with attribute-mediated dependencies. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2005)*, 2005.

[103] Parag Singla and Pedro Domingos. Entity resolution with Markov Logic. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2006)*, 2006.

[104] Rion Snow, Daniel Jurafsky, and Andrew Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of COLING/ACL 2006*, 2006.

[105] S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Machine Learning*, 34(1–3):233–272, 1999.

[106] R. Sombatsrisomboon, Y. Matsuo, and M. Ishizuka. Acquisition of Hypernyms and Hyponyms from the WWW. In *Procs. of 2nd International Workshop on Active Mining*, Maebashi City, Japan, 2003.

[107] S. Staab and A. Maedche. Ontology engineering beyond the modeling of concepts and relations. In *Proceedings of ECAI*, 2000.

[108] P.J. Stone. General inquirer. In *http://www.wjh.harvard.edu/ inquirer/*.

[109] C. Sutton and A. McCallum. Collective segmentation and labeling of distant entities in information extraction. In *University of Massachusetts TR 04-49*, 2004.

[110] H. Takamura, T. Inui, and M. Okumura. Extracting semantic orientation of words using spin model. In *Proceedings of the ACL*, 2005.

[111] H. Takamura, T. Inui, and M. Okumura. Extracting semantic orientations of words using spin model. In *Procs. of ACL*, pages 133–141, 2005.

[112] K. Torisawa. An unsupervised learning method for commonsensical inference rules on events. In *CoLogNET-ElsNet Symposium*, 2003.

[113] P. Turney. Inference of Semantic Orientation from Association. In *CoRR cs. CL/0309034*, 2003.

[114] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the Twelfth European Conference on Machine Learning (ECML-2001)*, pages 491–502, Freiburg, Germany, 2001.

[115] P. D. Turney. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Procs. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 129–159, Philadelphia, Pennsylvania, 2002.

[116] P. D. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Procs. of ACL*, pages 417–424, 2002.

[117] P. D. Turney and M. Littman. Measuring Praise and Criticism: Inference of Semantic Orientation from Association. *ACM Transactions on Information Systems (TOIS)*, 21(4):315–346, 2003.

[118] O. Uryupina. Semi-supervised learning of geographical gazetteers from the internet. In *Proceedings of the HLT-NAACL 2003 Workshop on Analysis of Geographic References*, pages 18–25, 2003.

[119] O. Uryupina. Semi-Supervised Learning of Geographical References within Text. In *Procs. of the NAACL-03 Workshop on the Analysis of Geographic References*, pages 21–29, Edmonton, Canada, 2003.

[120] J. Volker, D. Vrandecic, and Y. Sure. Automatic evaluation of ontologies (AEON). In *Proceedings of ISWC*, 2005.

[121] Michael Wick, Aron Culotta, and Andrew McCallum. Learning field compatibilities to extract database records from unstructured text. In *EMNLP*, 2006.

[122] T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Procs. of HLT-EMNLP*, 2005.

[123] T. Wilson, J. Wiebe, and R. Hwa. Just how mad are you? finding strong and weak opinion clauses. In *Procs. of AAAI*, pages 761–769, 2004.

[124] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically mined common sense. In *AAAI*, 2005.

[125] A. Yates, S. Schoenmackers, and O. Etzioni. Detecting parser errors using Web-based semantic filters. In *Proceedings of the EMNLP*, 2006.

[126] J. Yi, T. Nasukawa, R. Bunescu, and W. Niblack. Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Procs. of ICDM*, pages 1073–1083, 2003.

[127] J.Angele Y.Sure and S.Staab. OntoEdit: Multifaceted Inferencing for Ontology Engineering. *Journal of Data Semantics*, 1(1):128–152, 2003.

# Appendix A
# CLASS EXTRACTION FROM THE WEB

## *A.1 Examples of Extracted Class Information*

Chapter 3 describes our work on extracting subclass and related class information from the Web - it also includes some examples of discovered subclasses and related classes. In the following, we list additional examples of extracted IS-A links for WordNet enrichment.

### *A.1.1 Subclass Extraction For WordNet Enrichment*

```
WordNet Concept: parasite_0          WordNet Concept: construction_4          WordNet Concept: carnivore_0
--------------------------------------------------------------------------------------------------------
[ 0.99 - 0.90 ]
--------------------------------------------------------------------------------------------------------
subclassOf(parasite:flies)           subclassOf(construction:drilling)     subclassOf(carnivore:wolves)
subclassOf(parasite:worms)                                                 subclassOf(carnivore:tigers)
subclassOf(parasite:tapeworms)                                             subclassOf(carnivore:snakes)
subclassOf(parasite:lice)                                                  subclassOf(carnivore:skunks)
subclassOf(parasite:ants)                                                  subclassOf(carnivore:raccoons)
subclassOf(parasite:roundworms)                                            subclassOf(carnivore:pumas)
subclassOf(parasite:protozoa)                                              subclassOf(carnivore:mink)
subclassOf(parasite:hookworms)                                             subclassOf(carnivore:lions)
subclassOf(parasite:giardia)                                               subclassOf(carnivore:jaguars)
subclassOf(parasite:beetles)                                               subclassOf(carnivore:jackals)
subclassOf(parasite:ticks)                                                 subclassOf(carnivore:hyenas)
subclassOf(parasite:bugs)                                                  subclassOf(carnivore:foxes)
subclassOf(parasite:mites)                                                 subclassOf(carnivore:dogs)
subclassOf(parasite:ringworms)                                             subclassOf(carnivore:coyotes)
subclassOf(parasite:amoebae)                                               subclassOf(carnivore:cheetahs)
subclassOf(parasite:heartworm)                                             subclassOf(carnivore:raptors)
subclassOf(parasite:fungi)
subclassOf(parasite:threadworms)
subclassOf(parasite:trypanosomes)
subclassOf(parasite:viruses)
subclassOf(parasite:malaria)
subclassOf(parasite:nematodes)
subclassOf(parasite:pinworms)
----------------------------------------------------------------------------------------------------------
[ 0.90 - 0.85 ]
----------------------------------------------------------------------------------------------------------
subclassOf(parasite:sponges)         subclassOf(construction:tunneling)      subclassOf(carnivore:wolverine)
subclassOf(parasite:crabs)           subclassOf(construction:excavation)     subclassOf(carnivore:vultures)
subclassOf(parasite:earwigs)         subclassOf(construction:installation)   subclassOf(carnivore:mongooses)
subclassOf(parasite:mice)            subclassOf(construction:paving)         subclassOf(carnivore:mites)
```

|  |  |  |
|---|---|---|
| subclassOf(parasite:whipworms) | subclassOf(construction:removal) | subclassOf(carnivore:lynx) |
| subclassOf(parasite:dinoflagellates) | subclassOf(construction:laying) | subclassOf(carnivore:leopard) |
| subclassOf(parasite:roaches) | subclassOf(construction:framing) | subclassOf(carnivore:felids) |
| subclassOf(parasite:rabies) | subclassOf(construction:dredging) | subclassOf(carnivore:falcons) |
| subclassOf(parasite:flatworms) | subclassOf(construction:clearing) | subclassOf(carnivore:cougars) |
|  | subclassOf(construction:quarrying) | subclassOf(carnivore:bobcats) |
|  | subclassOf(construction:blasting) |  |
|  | subclassOf(construction:filling) |  |
|  | subclassOf(construction:staging) |  |
|  | subclassOf(construction:erosion) |  |

# Appendix B

# ANNOTATING RELATIONS WITH META-PROPERTY AND DEPENDENCY LABELS

## *B.1 Templates of Useful Lexico-syntactic Patterns*

| **Symmetric** | **Pattern Templates** |
|---|---|
| yes | $pl(X) \vert enum(x, x')$ **are** $noPrep(R) \vert sg(noPrep(R)) \vert pl(noPrep(R))$ .\|; \|, |
| no | $R$ **(*) but\|and (*) the other way around\|viceversa\| reversely \| conversely \| reverse \| opposite \| (*)** $R$ |
| no | $R$ (*) **and** \|**but** $R$ **back** |

Table B.1: **Templates for Patterns Correlated with the Presence or Absence of Relation Symmetry For Target Relation** $R \subseteq X \times X$**.** Notation: *enum()* = enumeration pattern (including conjunctions), $x$,$x'$ = elements of $X$, $sg(), pl()$ = plural form, $noPrep()$ = function that removes the preposition at the end of the relation's name.

| **Transitive** | **Pattern Template** |
|---|---|
| yes | **in turn** $R$ |
| yes | $R$ $x_1$ **which\|and therefore\|thus\|because\|as \| since\|and so (*)** $R$ $x_2$ |
| yes | $R$ $x_1$ **which\|and therefore\|thus\|because\|as\| since\|and so (*)** $R'$ $x_2$ , *entails(R, R′)* |

Table B.2: **Templates For Patterns Correlated with Relation Transitivity For Target Relation** $R \subseteq X \times X$**.** Notation: $x$,$x_1$,$x_2$ = values in $X$, $entails(R, R')$ = $R'$ is entailed by $R$.

| **1-to-1** | **Pattern Template** |
|---|---|
| yes | $uniqueMod\ sg(R)\ sg(Y)\|instanceOf(Y)$ |
| no | $multipleMod\ sg(R)\ sg(Y)\|instanceOf(Y)$ |
| yes | $sg(X)\|instanceOf(X)\ R\ sg(Y)\|instanceOf(Y)$ |
| no | $pl(X)\|enum(x,x')\ R\ sg(Y)\|instanceOf(Y)$ |

Table B.3: **Templates for Patterns Correlated With the Presence or Absence of the 1-to-1 Property for Target Relation $R \subseteq X \times Y$.** Notation: *sg(), pl()* = singular/plural forms; $x, y$ = elements of $X, Y$; *uniqueMod, multipleMod* = n-grams that indicate *countability* (*e.g.*, "the" is an example of a modifier indicating uniqueness, "another" is an example of a modifier indicating multiplicity), *enum()* = enumeration. Note: The patterns in this table are used in addition to the patterns below, common to the 1-to-1 and functional properties.

| **Functional** | **Pattern Template** |
|---|---|
| yes | $R\ sg(Y)\|instanceOf(Y)$ |
| yes | $sg(Y)\|instanceOf(Y)$**'s** (*) *noPrep*$(R)$ |
| no | $R\ pl(Y)\|enum(y,y')$ |
| no | $pl(Y)\|enum(y,y')$**'s** (*) *noPrep*$(R)$ |

Table B.4: **Templates for Patterns Correlated With the Presence or Absence of the Functional Property for Target Relation $R \subseteq X \times Y$.**

| **Dependency Type** | **Pattern Templates** |
|---|---|
| Entailment | $R$ **(*) therefore** \| **thus** \| **and/or even** \| **(*) in turn** $R'$ |
| Entailment | $R$ **(*) because it** \| **before** \| **in order to** \| **on the way to** $R'$ |
| Transitive-through | **because**\| **since** \| $R$ **(*) which** \| **that (*)** $R'$ |

Table B.5: **Templates for Patterns Correlated With Entailment and the Transitive-Through Dependency.** Notation: $R, R'$ = relations of interest

# Appendix C
# MINING PRODUCT FEATURES AND CORRESPONDING OPINIONS FROM REVIEWS

## C.1 Review Sets, Features and Opinion Cluster Information

Chapter 5 describes our work on mining product features and corresponding opinions from product reviews. In the following, we describe the data used as well as the mined information in more detail, give examples of learned features and discovered opinions (and opinion clusters) and indicate where the product reviews can be found.

### C.1.1 Product Review Sets

Our main experimental results used the product reviews compiled by B.Liu and M.Hu, which are publicly available from their website (see [49] for details); additionally, we used a set of Hotel reviews and a set of Scanner reviews - the former were obtained with the help of Fetch.com from *tripadvisor.com* and the latter were collected (using a locally developed wrapper) from Amazon.com.

The Hotel and Scanner reviews will be available shortly from the OPINE project site:

OPINE **Project Site**: *http://www.cs.washington.edu/homes/amp/opine/*

The site also contains a demonstration of the system's summarization capabilities on a larger set of approximately 10,000 Hotel reviews obtained from Fetch.com as part of the CALO project [1].

### C.1.2 Explicit Features for the Hotel and Scanner Product Classes

The OPINE project site contains numerous examples of explicit features mined from Hotel reviews. OPINE was also used to identify features corresponding to numerous other products of interest to the CALO project - in the following, we list the features corresponding to the Scanner class (a product class for which a large set of reviews had been compiled).

**Explicit Features for the Scanner Product Class**

*Notation:* “Feature” = name of explicit feature, “Type” = feature type: part, property, related concept (“rel”), “Of” = indicates the concept to which the feature corresponds. The feature information is organized in a 2-column format.

```
-----------------------------------------------------------------------------------------
Product Class: Scanner
-----------------------------------------------------------------------------------------
| Feature: [] Of: [] Type: []  | Feature: [] Of: [] Type: []
-----------------------------------------------------------------------------------------
| Feature: scanner Of:  Type: | Feature: software Of: scanner Type: part
| Feature: scan Of: scanner Type: rel | Feature: image Of: scanner Type: rel
| Feature: photo Of: scanner Type: rel | Feature: picture Of: scanner Type: rel
| Feature: button Of: scanner Type: part | Feature: resolution Of: scanner Type: property
| Feature: price Of: scanner Type: property | Feature: driver Of: scanner Type: part
| Feature: color Of: scanner Type: property | Feature: size Of: scanner Type: property
| Feature: printer Of: scanner Type: part | Feature: window Of: scanner Type: part
| Feature: setting Of: scanner Type: property | Feature: interface Of: scanner Type: part
| Feature: speed Of: scanner Type: property | Feature: port Of: scanner Type: part
| Feature: support Of: scanner Type: rel | Feature: cable Of: scanner Type: part
| Feature: card Of: scanner Type: part | Feature: hardware Of: scanner Type: part
| Feature: connection Of: scanner Type: part | Feature: screen Of: scanner Type: part
| Feature: application Of: scanner Type: rel | Feature: glass Of: scanner Type: part
| Feature: fax Of: scanner Type: part | Feature: cover Of: scanner Type: part
| Feature: light Of: scanner Type: part | Feature: function Of: scanner Type: property
| Feature: case Of: scanner Type: part | Feature: lamp Of: scanner Type: part
| Feature: warranty Of: scanner Type: property | Feature: capability Of: scanner Type: property
| Feature: performance Of: scanner Type: property | Feature: panel Of: scanner Type: part
| Feature: footprint Of: scanner Type: part | Feature: document_feeder Of: scanner Type: part
| Feature: feeder Of: scanner Type: part | Feature: bed Of: scanner Type: part
| Feature: firmware Of: scanner Type: part | Feature: arm Of: scanner Type: part
| Feature: depth Of: scanner Type: property | Feature: manual Of: scanner Type: rel
| Feature: lens Of: scanner Type: part | Feature: body Of: scanner Type: part
| Feature: detector Of: scanner Type: part | Feature: life Of: scanner Type: property
| Feature: control_panel Of: scanner Type: part | Feature: field Of: scanner Type: part
| Feature: copy Of: scanner Type: rel | Feature: detector Of: scanner Type: part
| Feature: back Of: scanner Type: part | Feature: density Of: scanner Type: rel
| Feature: attachment Of: scanner Type: part | Feature: cord Of: scanner Type: part
| Feature: drum Of: scanner Type: part | Feature: base Of: scanner Type: part
| Feature: adaptor Of: scanner Type: part | Feature: bias Of: scanner Type: property
| Feature: input Of: scanner Type: part | Feature: head Of: scanner Type: part
| Feature: look Of: scanner Type: property | Feature: keyboard Of: scanner Type: part
| Feature: dialog Of: scanner Type: part | Feature: capture Of: scanner Type: part
| Feature: door Of: scanner Type: part | Feature: delay Of: scanner Type: property
| Feature: focus Of: scanner Type: property | Feature: carrier Of: scanner Type: part
| Feature: display Of: scanner Type: part | Feature: grain Of: scanner Type: property
| Feature: controller Of: scanner Type: part | Feature: holder Of: scanner Type: part
| Feature: id Of: scanner Type: part | Feature: camera Of: scanner Type: part
| Feature: facility Of: scanner Type: property | Feature: build Of: scanner Type: property
| Feature: framework Of: scanner Type: part | Feature: board Of: scanner Type: part
| Feature: fan Of: scanner Type: part | Feature: bar Of: scanner Type: part
| Feature: history Of: scanner Type: part | Feature: fit Of: scanner Type: part
| Feature: data Of: scanner Type: part | Feature: form Of: scanner Type: property
| Feature: class Of: scanner Type: property | Feature: make Of: scanner Type: property
| Feature: noise Of: scanner Type: property | Feature: adjustment_resolution Of: scanner Type: property
| Feature: hardware_resolution Of: scanner Type: property | Feature: horizontal_resolution Of: scanner Type: property
| Feature: image_resolution Of: scanner Type: property | Feature: optical_resolution Of: scanner Type: property
| Feature: output_resolution Of: scanner Type: property | Feature: physical_resolution Of: scanner Type: property
| Feature: preview_resolution Of: scanner Type: property | Feature: scan_resolution Of: scanner Type: property
| Feature: screen_resolution Of: scanner Type: property | Feature: shadow_resolution Of: scanner Type: property
| Feature: site_resolution Of: scanner Type: property | Feature: software_resolution Of: scanner Type: property
| Feature: ultra_resolution Of: scanner Type: property | Feature: vertical_resolution Of: scanner Type: property
| Feature: video_resolution Of: scanner Type: property | Feature: web_resolution Of: scanner Type: property
| Feature: application_software Of: scanner Type: part | Feature: brochure_software Of: scanner Type: part
| Feature: built-in_software Of: scanner Type: part | Feature: bundle_software Of: scanner Type: part
| Feature: calibration_software Of: scanner Type: part | Feature: character recognition_software Of: scanner Type: part
| Feature: character_software Of: scanner Type: part | Feature: communication_software Of: scanner Type: part
| Feature: consumer_software Of: scanner Type: part | Feature: control_panel_software Of: scanner Type: part
| Feature: creator_software Of: scanner Type: part | Feature: device_software Of: scanner Type: part
| Feature: director_software Of: scanner Type: part | Feature: disc_software Of: scanner Type: part
| Feature: document_management_software Of: scanner Type: part  | Feature: document_software Of: scanner Type: part
| Feature: editing_software Of: scanner Type: part | Feature: editor_software Of: scanner Type: part
| Feature: email_software Of: scanner Type: part | Feature: film_software Of: scanner Type: part
| Feature: grade_software Of: scanner Type: part | Feature: hardware_software Of: scanner Type: part
| Feature: image-editing_software Of: scanner Type: part | Feature: image_manipulation_software Of: scanner Type: part
| Feature: image_processing_software Of: scanner Type: part | Feature: image_recognition_software Of: scanner Type: part
| Feature: image_software Of: scanner Type: part | Feature: imaging_software Of: scanner Type: part
| Feature: indicator_software Of: scanner Type: part | Feature: interface_software Of: scanner Type: part
| Feature: knowledge_base_software Of: scanner Type: part | Feature: layout_software Of: scanner Type: part
| Feature: level_software Of: scanner Type: part | Feature: loading_software Of: scanner Type: part
| Feature: mail_software Of: scanner Type: part | Feature: main_software Of: scanner Type: part
| Feature: manufacturer_software Of: scanner Type: part | Feature: menu_software Of: scanner Type: part
| Feature: music_software Of: scanner Type: part | Feature: newsletter_software Of: scanner Type: part
| Feature: number crunching_software Of: scanner Type: part | Feature: ocr_software Of: scanner Type: part
| Feature: optical_software Of: scanner Type: part | Feature: panel_software Of: scanner Type: part
| Feature: parallel_software Of: scanner Type: part | Feature: particular_software_package Of: scanner Type: part
| Feature: photo-editing_software Of: scanner Type: part | Feature: photo-enhancing_software Of: scanner Type: part
| Feature: photo-manipulation_software Of: scanner Type: part | Feature: picture-organization_software Of: scanner Type: part
| Feature: preview_software Of: scanner Type: part | Feature: print_software Of: scanner Type: part
| Feature: processing_software Of: scanner Type: part | Feature: reference_software Of: scanner Type: part
| Feature: scan-management_software Of: scanner Type: part | Feature: scanner_software_driver Of: scanner Type: part
| Feature: scanning_software_side Of: scanner Type: part | Feature: scan_software Of: scanner Type: part
| Feature: scan_software_setting Of: scanner Type: part | Feature: scan_suite_software Of: scanner Type: part
| Feature: setup_software Of: scanner Type: part | Feature: site-creation_software Of: scanner Type: part
| Feature: site_software Of: scanner Type: part | Feature: software_application Of: software Type: part
| Feature: software_assistance Of: software Type: rel | Feature: software_character Of: software Type: property
| Feature: software_check Of: software Type: rel | Feature: software_component Of: software Type: part
| Feature: software_control Of: software Type: property | Feature: software_driver Of: software Type: part
| Feature: software_ease Of: software Type: property | Feature: software_enhancement Of: software Type: property
| Feature: software_feature Of: software Type: part | Feature: software_icon Of: software Type: part
| Feature: software_installation Of: software Type: part | Feature: software_installing Of: software Type: part
| Feature: software_instruction Of: software Type: part | Feature: software_program Of: software Type: part
| Feature: software_quality Of: software Type: property | Feature: software_setting Of: software Type: property
| Feature: software_title Of: software Type: property | Feature: software_upgrade Of: software Type: part
| Feature: software_usability Of: software Type: property | Feature: software_vendor Of: software Type: rel
```

### *C.1.3 Opinion Examples*

In the following, we list examples of identified opinions with *dominant* positive and negative semantic orientation labels (the domain is that of Hotel reviews). We then give examples of contexts that influence the semantic orientation of words such as *cold*, *hot*, *small* and *casual*.

```
Examples of Positive Opinion Words                          Examples of Negative Opinion Words
-----------------------------------------------------       ---------------------------------------------------
possible intimate finest luxurious adequate thrilling       unsafe preserved hate ordinary painful
prefer sumptuous intercontinental brisk bargain             unable difficult rainy empty terrible
advantage pleased necessary fantastic durable               inadequate ugly moldy kids unsubstantial
reliable colorful cute undamaged eager                      unclean long sharp colorless abstract
marvelous free airy calm rested                             unpopular impure nonaged malicious retired
important discerning delicious casual weak                  unresponsive stingy incomplete obsequious
ready dry available low-pitched friendly                    contaminating corrupt stale alcoholic cautionary
traditional super gigantic tasteful domestic                busy imperfect dull afraid lucky
at_least gem viewable stylish love space                    unoriginal forced inappropriate incorrect individual
quiet short good welcoming high-end                         soiled misguided trick outdated adjacent
specific informal complete stocked gorgeous                 nonmodern depressed mediocre dark unequipped
best generous concrete timely complimentary                 unpleasant nominal busted unhelpful colourless
wireless popular ideal interesting unique                   disgusting evil formal insecure supernatural
European-style original modern heavenly effusive            bound unavailable off unprotected mediate
bonus favorite lucious unmistakeable impeccable             irresponsible noncrucial unreal wired
```

```
|Feature|Opinion|Semantic Orientation Label|Review Sentence|
bathroom_shower_temperature*cold*-*Cold showers are not fun .*
bathroom_shower_temperature*cold*-*One morning I couldnt figure out how to get the hot water
                                   to work so I had to take a cold shower .*
bathroom_shower_temperature*cold*-*I had to take a cold shower because the hot water was out
                                   on the 6th floor .*
atmosphere_temperature*cold*-*On first impressions, the atmosphere was quite cold and un-welcoming
                                   from the deskperson and we were horrified to find that we had to
                                   carry our extremely large bags up three flights of steep stairs .*
breakfast_temperature*cold*-*The buffet breakfast that was included was cold and bland, if not just
                                   generally depressing for the lack of options .*
breakfast_temperature*cold*-*The Fettucini Alfredo was awful and the breakfast I had at the cafe was cold .*
breakfast_temperature*cold*+*Cold breakfast is included is a bonus and much appreciated .*
breakfast_temperature*cold*+*The Inn serves a cold breakfast that is ok .*
breakfast_temperature*cold*+*Our NY1 rate included a full, cold buffet breakfast which included some
                                   of the best bagels I have tasted .*
room_temperature*cold*+*great views and ice cold - it was hot so the cold room was wonderful .*
room_temperature*cold*-*Once in the bitter cold room, my husband tried to figure out what
                                   the problem was with the heat .*
room_temperature*hot*-*The lobby is done out poshly, but I was on floor 14 and I thought Id gone to hell ,
                                   hot and humid , noisy , tiny room, dirty bathrooms, construction from 8am
                                   every morning .*
room_temperature*hot*-* Hot, small, stuffy, noisy room .*
bathroom_shower_temperature*hot*+*At least the dribbling shower was hot .*
bathroom_shower_temperature*hot*-*When taking a hot shower, it became difficult to breathe .*
food_temperature*hot*+*The breakfasts at this hotel were most enjoyable, with a great selection of hot
                                  foods, cereals, fruits,and many other things.*
atmosphere_quality*casual*+*Very upscale and luxurious but it maintains a casual atmosphere, this is
                                  leisure traveling at its best.*
```

```
restaurant_quality*casual*+*We enjoyed good service in the casual restaurant, and wonderful cocktails
                                 in the bar .*
restaurant_quality*casual*-*The restaurant was casual, the decor tasteless and the food
                                 was at best average.*
room_size*small*+*A small room with fabulous breakfast every morning .*
room_size*small*+* Room is small but not cramped, and very, very clean .*
room_size*small*+*Good clean small room in a vibrant neighborhood with good local restaurants .*
room_size*small*-*The room was small but it was pretty clean .*
room_size*small*-*The staff are indifferent, the lobby feels like a taxi-cab booking office and the rooms
                               are small, uncomfortable and stale .*
```

### *C.1.4 Opinion Clusters and Implicit Features*

In the following, we give examples of opinion clusters and corresponding implicit features learned from Hotel and Scanner data - for the purposes of the CALO project, this data is being used as an initial lexicon for many new product domains with good initial results.

```
-----------------------------------------------------------------------------------------
Cluster Name|Cluster Elements
-----------------------------------------------------------------------------------------
NONAME|crowded|uncrowded|packed
NONAME|discerning|undiscerning
NONAME|downtown|uptown
NONAME|earthly|divine|heavenly
NONAME|factory-made|homemade
NONAME|inefficient|efficient
NONAME|intercontinental|continental
NONAME|known|unknown
NONAME|looseleaf|bound
NONAME|lost|missing|found
NONAME|lost|saved
NONAME|lost|won
NONAME|lyric|dramatic
NONAME|marked|unmarked
NONAME|maximal|minimal|nominal|maximum
NONAME|misused|used
NONAME|nonprofessional|professional
NONAME|off|on
NONAME|professional|unprofessional
NONAME|steep|gradual|sloping|
NONAME|stocked|unequipped|equipped|unfurnished|furnished
NONAME|unwritten|spoken|written
NONAME|welcome|unwelcome
NONAME|wired|bound
NONAME|wireless|wired
adequacy|inadequate|adequate|poor|decent
age|stale|antique|old|recent|fresh|used|tired|new|retired|aged|young|ancient|preserved|aged
artificiality|inorganic|artificial|organic
attention|solicitous|attentive|dreamy|inattentive|heed|thoughtless|heedless|unheeding
be_casual|casual|formal|informal|nominal
be_frivolous|frivolous|light|real|serious
be_generic|generic|popular|special|plain|broad|specific|general
be_open|open|close
```

```
beauty|ugly|gorgeous|unnatural|beautiful|lovely
boldness|amazing|spectacular|awful|unimpressive|dramatic|stunning|impressive|outstanding
calmness|peaceful|stormy|calm|serene|unpeaceful
centrality|central|peripheral
clarity|clear|unclear|intuitive|understandable
cleanliness|spotless|clean|soiled|dirty|unclean|dusty|greasy|impeccable
coarseness|coarse|indecent|decent
color_property|bright|colorless|colourless|drab|colorful|uncoloured|uncolored
color|white|blue|red|orange|green|purple|black|yellow|pink|brown
comfortableness|rich|comfortable|affluent||poor
comfort|comfy|painful|comfortable|uncomfortable|cozy
commonness|coarse|common
completeness|broken|full|incomplete|hearty|uncomplete|complete
complexity|complex|simple|compound
convenience|inconvenient|convenient
correctness|straight|correct|wrong|incorrect|false|misguided|appropriate|inappropriate
credibility|credible|incredible|believable
curliness|straight|curly
damage|broken|busted|torn|undamaged|damaged
depth|shallow|deep
discreetness|discreet|indiscreet
distance|close|immediate|adjacent|deep|near|distant|far|nearby
ease|painless|painful|tough|hard|easy|difficult
elegance|inelegant|elegant|deluxe
evil|malevolent|evil|wicked
excitement|exciting|unexciting|thrilling
expectedness|unexpected|expected
falseness|hollow|false
fear|fearless|unafraid(p)|unafraid|afraid
fight|competitive|noncompetitive
foreignness|native|foreign|domestic|strange
formality|elegant|formal
fortune|fortunate|unfortunate
frequency|frequent|infrequent
freshness|stale|moldy|fresh
friendliness|unfriendly|friendly|warm
functionality|functional
fun|playful|serious
generosity|generous|ungenerous|stingy
handiness|available|unavailable|
handiness|free|unfree|available|bound|available|unavailable
happiness|pleased|displeased|delighted|happy|unhappy|dysphoric|distressed
helpfulness|unhelpful|unaccommodating|accommodating|helpful
historicalness|synchronic|descriptive|historical
idleness|busy|idle|responsible|irresponsible
immediacy|mediate|immediate
importance|important|unimportant|significant|insignificant|crucial|noncrucial|minor|major
inertness|neutral|indifferent
intelligence|bright|smart|stupid
intensity|intensive|extensive
interest|dull|uninteresting|interesting
intimacy|public|close|private
irreality|uncolored|natural|unreal|artificial|near|false
irregularity|irregular|atypical
kindness|generous|ungenerous|kind|meanspirited|malevolent|malicious
level|low|high
light|dark|bright|light|dim|depressing
```

```
likeness|like|similar|same|different
luck|unlucky|lucky|luckless
lusciousness|delightful|delicious
narrow-mindedness|broad-minded
naturalness|unnatural|natural|forced|artificial
necessity|unneeded|necessary|extra|unnecessary|needed
noise|quiet|dreamy|noisy|loud|dull|deafening|silent
pitch|low-pitched
pleasantness|enjoyable|painful|unhappy|pleasant|unpleasant|gracious|weird|charming|nasty|awful|lousy
politeness|polite|gracious|courteous|rude|impolite|discourteous|ungracious
popularity|popular|hot|unpopular|favorite
possibility|possible|impossible
power|not_able|unable|able|strong|impotent|weak
price|overpriced|cheap|bargain|expensive|free|complimentary|uncomplimentary
purity|dirty|impure|pure
quality|great|tough|satisfactory|lousy|poor|mediocre|bad|negative|good|solid|superb|unsatisfactory
             |finest|good|best|worst|rich|fine|off|fabulous|fantastic|marvelous|outstanding|special
             |average|phenomenal|terrific|uncommon|common|extraordinary|ordinary|perfect|imperfect
             |worsened|better|worse|coarse|disgusting|wicked|nice
quantity|abundant|rich|scarce|extensive
readiness|unready|ready
reality|abstract|concrete|ideal|real|unreal
reasonableness|reasonable|unreasonable
recency|modern|nonmodern|recent|antique|trendy|unfashionable|fashionable|quaint|old-fashioned|dated
                         |stylish|cool|original|fresh|unoriginal|tired
regularity|frequent|regular|infrequent
responsiveness|responsive|unresponsive
richness|plush|sumptuous|deluxe|luxurious
roominess|spacious|roomy|incommodious|cramped|tiny
safety|secure|safe|unsafe|insecure|protected|unprotected
separateness|public|separate|idiosyncratic|individual|common
separation|joint|separate
sharpness|sharp|dull
sign|electroneutral|neutral|positive|negative|electronegative
size|spacious|wide|large|little|extensive|deep|small|super|gigantic|big|broad|huge|microscopic|tiny
smartness|smart|fresh
softness|soft|hard
solidity|broken|unbroken|solid|hollow
speciousness|in_poor_taste(p)|tasteless|cheap|tasteful|flashy
speed|high-speed|quick|immediate|fast|slow|sluggish
state|concrete|solid|gaseous|liquid
substantiality|insubstantial|airy|unreal|real|unsubstantial
surprisingness|unexpected|forced
temperature|warm|hot|cold|cool|chilly|balmy
thinness|thin|thick
timing|past|historic|future|historical
traditionalism|traditional|untraditional|nontraditional
truth|true|wrong|real|false
typicality|untypical|typical|true|atypical
usualness|ordinary|popular|unusual|special|average|general|common|uncommon
         |quaint|familiar|strange|crazy|weird|unfamiliar|unique
warmth|warm|tender
wetness|dry|damp|wet|rainy
width|narrow|wide|broad
willingness|unwilling|willing
```

# Appendix D
# ASSESSING AND MINING COMMONSENSE KNOWLEDGE FROM THE WEB

## *D.1 Examples of Mined and Assessed Commonsense Information*

Chapter 6 describes our efforts to assess and mine commonsense facts from the Web. The data used in the state recognition experiments described in Chapter 6 will soon be available on the Web, as discussed below.

- The *assessed version of existent OMICS knowledge* will soon be available at

  *http://www.cs.washington.edu/homes/amp/omics/.*

  The OMICS relations of interest are: *ContextActionObject*, *Cause*, *Desire*, *People*, *Response* and *StateChange*.

- The *OMICS-type knowledge* mined from the Web will soon be available at

  *http://www.cs.washington.edu:/homes/amp/omics/minedData/.*

# VITA

Ana-Maria Popescu was born in Bucharest, Romania and currently resides in Seattle, Washington. She earned a Bachelor of Science degree in Mathematics & Computer Science at Brown University (1999) and a Masters degree in Computer Science at University of Washington (2001). In February 2007 she earned a Doctor of Philosophy at the University of Washington in Computer Science.