Leveraging Knowledge Bases in Web Text Processing

Thomas Lin

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

Doctor of Philosophy

University of Washington

2012

Reading Committee:

Oren Etzioni, Co-Chair

Mausam, Co-Chair

Patrick Pantel

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Leveraging Knowledge Bases in Web Text Processing

Thomas Lin

Co-Chairs of the Supervisory Committee:
Professor Oren Etzioni
Computer Science and Engineering

Research Assistant Professor Mausam
Computer Science and Engineering

The Web contains more text than any other source in human history, and continues to expand rapidly. Computer algorithms to process and extract knowledge from Web text have the potential not only to improve Web search, but also to collect a sizable fraction of human knowledge and use it to enable smarter artificial intelligence. To scale to the size and diversity of the Web, many Web text processing algorithms use domain-independent statistical approaches, rather than limiting their processing to any fixed ontologies or sets of domains.

While traditional knowledge bases (KBs) had limited coverage of general knowledge, the last few years have seen the rapid rise of new KBs like Freebase and Wikipedia that now cover millions of general interest topics. While these KBs still do not cover the full diversity of the Web, this thesis demonstrates that they are now close enough that there are ways to effectively leverage them in domain-independent Web text processing. It presents and empirically verifies how these KBs can be used to filter uninteresting Web extractions, enhance understanding and usability of both extracted relations and extracted entities, and even power new functionality for Web search. The effective integration of KBs with automated Web text processing brings us closer toward realizing the potential of Web text.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ARG1: The *subject* argument in a (*subject*, *relation*, *object*) binary relational extraction.

ARG2: The *object* argument in a (*subject*, *relation*, *object*) binary relational extraction.

COLLECTIVE CONTEXT: A technique used by our entity linker which can combine evidence from multiple Web documents when assigning a link.

EM: Expectation Maximization is an iterative method for estimating parameters in statistical models.

ENTITY: A coherent concept. Wikipedia and Freebase contain information about millions of entities.

ENTITY LINKING: Associating text strings with their corresponding entries in an entity store.

FREEBASE: A public entity repository, available at http://www.freebase.com.

FUNCTIONALITY: A relation is *functional* if it maps each first argument to at most one second argument.

IE: Information Extraction refers to extracting knowledge from text sources.

INLINK: The inlink count of a Wikipedia entity is the number of articles in Wikipedia whose text contains hyperlinks to the entity's Wikipedia page.

INLINK RATIO: A ratio which can detect systematic linking errors by finding entities that are linked to at a different rate than expected.

KB: Knowledge Bases are large repositories of information. The primary knowledge bases we use are Wikipedia and Freebase.

LINK AMBIGUITY: When linking a noun phrase, link ambiguity corresponds to whether the best candidate is clearly better than all other candidates.

LINK SCORE: Link score corresponds to likely correctness of an entity link.

MUTUAL EXCLUSION: Mutually exclusive items cannot co-exist. For example, *color* type and *river* type are mutually exclusive because no entity is both a color and a river.

NER: Named Entity Recognition is an NLP subtask that locates and classifies named entities (typically *people*, *organizations* and *locations*) in text.

NLP: Natural Language Processing is the field of computer science studying how computers can interact with information expressed in human languages (*e.g.*, English).

OPEN IE: Open Information Extraction, a relation-independent form of IE.

RELATION WEIGHT: The probability that two random elements in the domain of the relation will share at least one type.

REVERB: An Open Information Extraction system [37].

TEXTRUNNER: An Open Information Extraction system [4].

TFIDF: Term Frequency-Inverse Document Frequency is a weight that evaluates how important a word is to a document in a corpus [104].

UNLINKABLE NOUN PHRASE: A noun phrase which cannot be entity linked because it does not reference any entity present in the KB.

WIKIPEDIA: An online encyclopedia, currently covering 3-4 million topics.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to his invaluable advisors, committee members, paper co-authors, and those that have helped along the way, including Oren Etzioni, Mausam, Patrick Pantel, James Fogarty, Meliha Yetisgen-Yildiz, Lindsay Michimoto, Stephen Soderland, Patrick Allen, Niranjan Balasubramanian, Jeff Huang, Tony Fader, Alan Ritter, Janara Christensen, Robert Bart, Ryan Oman, Michael Schmitz, Stefan Schoenmackers, Yoav Artzi, Michele Banko, Ana-Maria Popescu, Dan Weld, Xiao Ling, Raphael Hoffmann, Fei Wu, Congle Zhang, Wei Xu, YongChul Kwon, Cynthia Matuszek, Hoifung Poon, Michael Gamon, Yu Chen, Susan Dumais, Ryen White, Omar Alonso, Ariel Fuxman and Anitha Kannan. The author would also like to thank NDSEG, Yahoo, and the Turing Center for financial support.

DEDICATION

To my family.

Chapter 1

INTRODUCTION

The Web [8] now contains more text than any other source in human history, and continues to expand rapidly. Vast quantities of information can be found within its many billions of pages, which cover company and personal homepages, forum and social postings, encyclopedia and news articles, product and media catalogs, and much more. Computer algorithms for automatically reading, understanding, and cataloging this knowledge promise both to enable better knowledge-based tools for users, and also to provide AI programs with more background knowledge than previously possible [64, 114].

To scale to the size and diversity of the Web, many recent techniques have eschewed manual resources in favor of domain-independent statistical approaches [6]. One key example is Open Information Extraction (Open IE). Open IE systems [4, 34, 111] learn general models of how relations are expressed in text, in order to extract facts covering all varieties of topics and relations. The state-of-the-art ReVerb Open IE system [37] recently extracted over one billion binary extractions such as "(Clinton, was born in, 1946)" and "(tourists, visited, London)" from the Web.

While a vast collection of text extractions can enable automated question answering at a superficial level (*e.g.*, "list everything born in 1946"), it falls short of *understanding* at the level needed to support many applications. For example, does the string "Clinton" refer to a *person*, a *place*, or something else? What does the relation phrase "was born in" mean? Can one be "born in" multiple years? In some domains, matching against manually compiled knowledge resources (*e.g.*, SNOMED [93] for medical or Tipster Gazetteer for geographical) could support this, but the Web covers a vast array of topics, many of which have no such dedicated resources.

Recent years have seen the emergence and rapid growth of general-coverage knowledge bases such as Freebase [11] and Wikipedia [126] that offer significant coverage across much of

general knowledge. While these resources still fall short of covering all concepts mentioned on the Web [69, 123], they may have wide enough coverage now to start incorporating into open domain Web text processing. At the same time, a vital concern when integrating fixed resources into domain-independent methods is that such integration must not limit the scope of knowledge processed to only the vocabulary of the resource. Otherwise, generality is lost as the system becomes restricted to only predefined sets of relations (as in NELL [18] and PROSPERA [80]), or predefined sets of entities (as in Wikify! [77] and GLOW [98]).

In this thesis we investigate the following hypothesis:

> Domain-independent Web text processing techniques can leverage large, general-coverage knowledge bases to better understand and apply the diverse information expressed in Web text.

We validate this hypothesis by exploring several limitations faced by current domain-independent Web text processing techniques, and then designing and evaluating techniques to address these challenges and enable new functionality through the use of knowledge bases. First, we explore the challenge of improving general extraction usefulness. Then, we examine in turn how knowledge bases can help to achieve deeper understanding of *extracted relation phrases* and *extracted arguments*. Finally, we demonstrate an extension of this work to a common Web search scenario. We next describe each of these in further detail.

1.1 Section Overviews

A primary challenge with Open IE from Web text is that in addition to useful extractions such as "(Clinton, was born in, 1946)," meaningless extractions such as "(Clinton, was born in, that month)" are also frequently extracted [65, 67]. We found that less than half of top results from the TEXTRUNNER Open IE system [4] were judged to be interesting by people. In Chapter 2, we examine whether knowledge bases can help address this problem. We leverage an idea that *Wikipedia Infoboxes* are likely to contain interesting information to train a classifier for predicting extraction interestingness. When used as a filter, this classifier produced a 54% relative increase in the fraction of interesting TEXTRUNNER results. The

classifier outperforms other methods we evaluated including one based on TF-IDF [104], and it can be applied to all Web extractions without restricting vocabulary because it uses only unlexicalized features.

To more deeply understand extractions involves learning more about both extracted relations and extracted arguments. One challenge with relation phrases is determining *properties* they can hold [66], such as the *functionality* property [68]. A relation is functional if it maps each *subject* argument to at most one *object* argument. The relation "was born in (city)" is functional because each person is born in only one city. Meanwhile, the relation "visited (city)" is not functional because one can visit multiple cities. Accurate functionality detection benefits numerous tasks such as contradiction detection [100], quantifier scope disambiguation [117] and synonym resolution [130]. In Chapter 3, we show how data from the Freebase knowledge base can be effectively integrated when learning general relation properties for relation phrases. On the task of functionality detection for arbitrary Web relation phrases, our system integrating Freebase data achieves 33.5% increased AUC compared to a statistical method based on KL divergence [61], and also outperformed the existing state-of-the-art system [100].

In addition to relation phrases, extractions from Web text also contain arguments. The next step is to better understand these extraction arguments. When we see "(Clinton, was born in, 1946)," does "Clinton" refer to a *person*, a *place*, or something else? In Chapter 4, we begin by studying how Web extraction arguments can be efficiently linked to Wikipedia entries. In this case, linking "Clinton" to Wikipedia's "Bill Clinton" entry can disambiguate the reference from other Clintons, and provide the typing information that "Clinton" refers to a *person* and a *politician*. We further observe that many Web extraction arguments refer to entities not prominent enough to be in Wikipedia. To handle these, we design and implement techniques to *detect* and *assign types* to all of the *unlinkable* (non-Wikipedia) entities in our data, by leveraging the large number of entities that we could link. While studies [81, 133] have observed that the Web contains many such unlinkable entities, this work is the first that attempts to make them useful rather than just separate them out. Our system achieves significantly higher performance than baselines in both detecting and typing the unlinkable entities, and enables greater recall in tasks such as typed question

answering [16] and inference [107].

Beyond extraction, use of large knowledge bases can also benefit other open domain Web text scenarios such as Web search. Chapter 5 shows that if leveraging KBs allows us to *detect* and *type* entities within search queries, then this further enables an ability to infer the *actions* that users have in mind when issuing those queries. We design and implement a system where given a search query containing an entity, we automatically infer relevant actions that the user is likely to be interested in taking. If they query for "Jetbeam flashlight," then the system might suggest "read reviews" and "buy online." While search engines [12] apply statistical techniques to keywords to handle the size and diversity of the Web, the addition of KBs is necessary in order to enable experiences such as this one. The ability to *detect* entities enables a basic version of our *actions*, and the further ability to *type* the entities increases the nDCG metric by over 20%, and allows the system to recommend actions that human judges rated highly in an evaluation.

As individual contributions, we introduce new techniques that improve upon the appropriate baselines and enable compelling new functionality in a number of tasks that we examined. As an overall contribution, the techniques we devise and demonstrate suggest that large knowledge bases can now be effectively leveraged to benefit open-domain Web text processing tasks. Chapter 6 describes several promising avenues for continued future work, and Chapter 7 presents concluding remarks.

Chapter 2

IDENTIFYING INTERESTING WEB EXTRACTIONS

Information Extraction (IE) is one approach to building knowledge repositories by extracting knowledge from text. Open IE systems such as TextRunner [4] are able to extract hundreds of millions of assertions from Web text. However, because of imperfections in extraction technology and the noisy nature of Web text, Open IE systems return a mix of both useful, informative facts (*e.g.*, "the FDA banned Ephedra") and less informative statements (*e.g.*, "the FDA banned products").

This chapter begins by introducing Open IE from Web text, how it works, and also two state-of-the-art Open IE systems. It then proposes several models of what attributes make extractions useful and *interesting*, presents our implementation of these models as filters to filter out uninteresting extractions, and reports on measurements of their efficacy on a sample of queries. The primary technical contributions of this chapter are to:

1. Develop several practical models of *interestingness* that are informed by previous work and theories. When implemented as filters, they offer substantial improvements over the standard Open IE technique of sorting assertions by frequency. Our highest precision model leverages general knowledge available from the Wikipedia knowledge base.

2. Utilize a machine-learning method that combines all the models to filter out uninteresting assertions resulting from extraction.

3. Report on the first study of interestingness in extraction. For this study we compare the efficacy of our models to each other and the TextRunner baseline. Among other findings, we show that our filtering significantly improves the fraction of interesting results contained within TextRunner's top thirty results from 41.6% interesting to 64.1% interesting.

2.1 Open Information Extraction from the Web

Information extraction (IE) is a subfield of natural language processing that seeks to obtain structured information from unstructured text. IE can be used to automate the tedious and error prone process of collecting facts from the Web. While some IE systems [56, 99] can only operate over small, prespecified sets of relations, *Open* Information Extraction systems are designed to capture all relations in text, while staying fast and operating in running time independent of the number of relations. Open IE systems extract all knowledge from Web text by first collecting all the English sentences found over millions of Web pages, and then checking each sentence individually for knowledge to extract. Given the sentence "Diet Coke contains aspartame instead of sugar" from an online discussion forum, an Open IE system could extract the assertion ("Diet Coke", "contains", "aspartame"). This extraction would assert that a "Diet Coke" entity holds a "contains" relation with an "aspartame" entity. Repeating this process for millions of sentences builds together a vast collection of facts that can be used in later applications.

The most widely-used Open IE systems are TextRunner [4] and ReVerb [37]. TextRunner was developed in 2007, and has implementations using Naive Bayes [4] and Conditional Random Fields [5]. From each sentence, TextRunner first identifies all the noun phrases using a noun phrase chunker. It then examines the sequence of words between each pair of noun phrases, and tries to determine what relationship, if any, holds between them. The Naive Bayes version does this by first heuristically eliminating any non-essential words in the sequence, arriving at a relation phrase, and then using a Naive Bayes classifier to judge whether the resulting (noun phrase, relation phrase, noun phrase) candidate tuple expresses a good extraction. In the earlier example, TextRunner would first identify that the sentence contains noun phrases "Diet Coke," "aspartame," and "sugar." It would then identify "contains" as the relation between "Diet Coke" and "aspartame" to produce the ("Diet Coke", "contains", "aspartame") extraction.

The ReVerb extractor was developed in 2011, and starts by finding relations rather than noun phrases. When given a sentence, ReVerb traverses the sentence to identify any relation phrases that satisfy a number of general syntactic and lexical constraints.

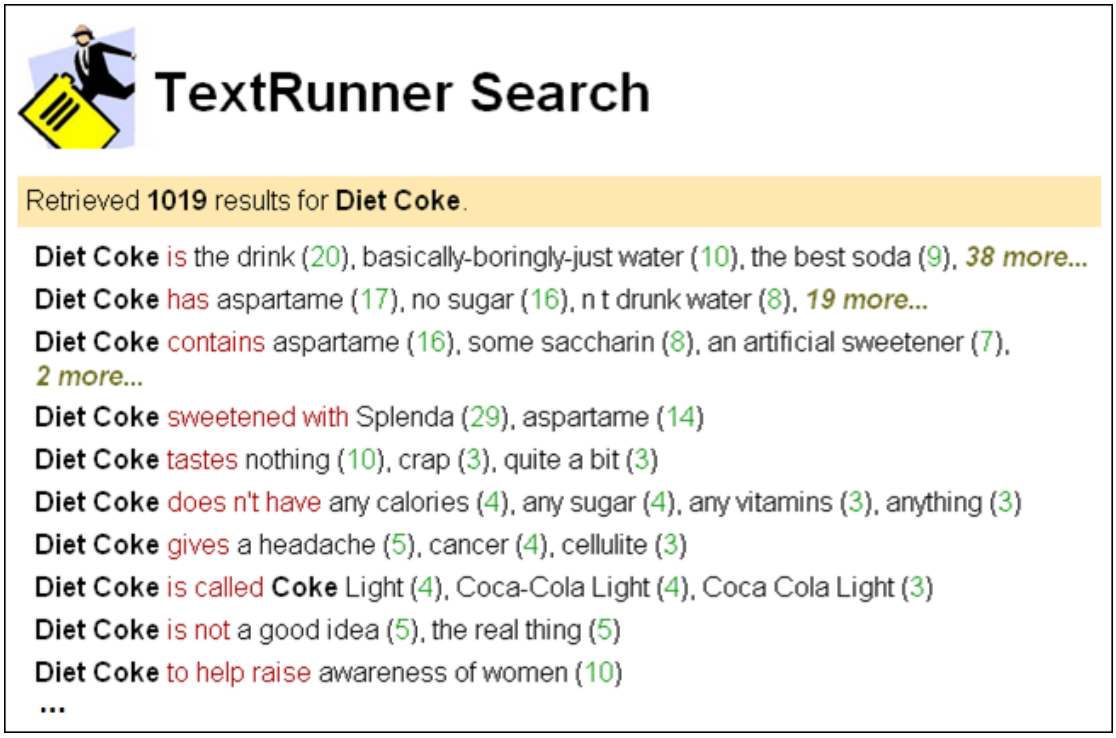

Figure 2.1: **Web Extractions:** After extracting millions of facts from the Web, Open IE systems can organize this information to support user queries.

When a valid relation phrase is identified, it then searches the rest of the sentence for the corresponding argument noun phrases. Finally, a logistic regression classifier scores the validity of the overall extraction. In the Diet Coke example, ReVerb would first identify that the sentence contains a relation phrase "contains." It would then locate "Diet Coke" and "aspartame" as the arguments of this relation to produce the final extraction.

TextRunner has been run on 500 million high-quality Web pages from Google to yield over 800 million extractions, and ReVerb has been run on 500 million Web pages from the ClueWeb09 corpus to yield over 1 billion extractions. The extractions were then indexed in Lucene to facilitate easy querying by entity and/or relationship. For example, if a person is interested in learning more about Diet Coke, they can issue this query to TextRunner and get back a list of extracted facts, as shown in Figure 2.1. TextRunner and ReVerb can also interpret and answer simple queries such as "What contains aspartame?" and hone

in on such answers as *"Trident gum"* and *"Diet Coke"* to free people from sifting through many Web pages to find the desired answers.

The first several chapters of this thesis focus on how the output extraction collections from Open IE on Web text can be improved and better understood by leveraging knowledge bases. The research conducted before ReVerb was developed (Chapter 2 and Chapter 3) uses primarily TextRunner data, while the research conducted after ReVerb was developed (Chapter 4) uses ReVerb data because ReVerb has generally better performance. The challenges explored and addressed in this thesis are common to both systems, and more generally, to many domain-independent Web text processing systems.

As shown in Figure 2.1, TextRunner results are returned ranked by frequency. The numbers in parenthesis are the numbers of times each extraction was observed. TextRunner ranks results by frequency because, all other things being equal, extractions that appear more frequently on Web pages are more likely to be correct [27]. However, this technique also yields many vague or otherwise uninteresting assertions. While extractions such as ("Diet Coke", "contains", "aspartame") and ("Diet Coke", "is called", "Coke Light") are useful, some uninformative extractions such as ("Diet Coke", "is not", "a good idea") also have relatively high frequency. Experiments presented in this chapter show that people find 58.4% of the thirty top-ranked answers returned by TextRunner to be uninformative and not interesting.

2.2 What's Interesting?

Extraction engines could be improved by filtering based on models of which extracted assertions are of interest and which are not. Figure 2.2 outlines this idea. Of course, the notion of interestingness is subjective, personal, and context specific. Nevertheless, any system that returns ranked results, from Google to TextRunner, either implicitly or explicitly utilizes a model of what is interesting in its ranking function.

2.2.1 Related Work in Interestingness

The general concept of using interestingness as a metric has value and applicability to a wide range of domains. For instance, Flickr recently launched a new feature for identifying

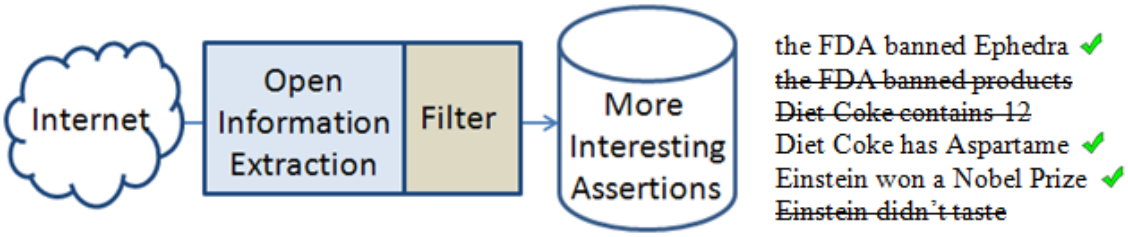

Figure 2.2: **Filtering Architecture:** Filtering the output of Open IE enables it to better focus on extracted assertions that are more *interesting*.

"Interestingness" in photos on its site.[1] The Flickr notion is based on social feedback such as click data and comments, supporting the idea that people care about what's interesting and leave indirect clues to where interesting content can be found. We use a similar concept later on in Section 2.3.4 by learning from how people populate Wikipedia infoboxes.

Similarly, automated mathematical discovery programs require a notion of interestingness in order to identify which potential conjectures and concepts will be of interest to people. Colton and Bundy's survey [22] identified several key concepts that these programs tended to use in deciding what would be interesting, including plausibility, novelty, surprisingness, comprehensibility and complexity. Liu *et al.* [73] found that unexpected database association rules are more interesting to users.

From psychology there are several theories of interestingness such as *complexity*, *novelty*, *uncertainty*, and *conflict* [113]. There has also been research into what attributes make *text* more interesting. It is important that text be the right level of complexity. Sentences with concrete words were found to be more interesting than abstract sentences [103]. Texts that are more coherent and easier to comprehend are more interesting [108]. Prior knowledge in the subject generally increases interest. These ideas help inform some of our classifier features later.

[1] http://www.flickr.com/explore/interesting/

2.3 Modeling Interestingness

2.3.1 Computational Formalizations

This section describes the problem of formalizing interestingness, then introduces three practical models for identifying interesting assertions. For each model, we first present an intuition behind characteristics that can make assertions interesting. We then operationalize those characteristics so that we can express them algorithmically.

We want to capture the interesting assertions, but what exactly does this mean? At the most general level, we define interesting assertions to be those that a person may find useful or engaging. For any particular query (*e.g.*, "Einstein"), the extent to which possible assertions are interesting may vary greatly. A good set of results might, for example, include a mix of biographical facts like "Einstein was born in Germany" and other interesting facts like "Einstein's favorite color was blue". On the other hand, "Einstein turned 15" or "Einstein wrote the paper" would be less interesting because they express little useful information.

In a discussion of what is interesting, personalization is one approach to consider. Different people will find different topics to be interesting. We consider personalized notions of interesting to be a future direction, but currently focus on what characteristics make an assertion broadly interesting to a variety of people.

2.3.2 Specific Assertions

One quality of interesting assertions is that they tend to provide more specific information. For example, "Albert Einstein taught at Princeton University" is more interesting than "Albert Einstein taught at a university" because identifying Princeton as the university is informative. We hypothesize this is one characteristic that can make assertions interesting more broadly in TextRunner.

To operationalize this quality, we define a *specific* assertion as an assertion that either relates multiple proper nouns or an assertion that contains a year. If an assertion relates multiple proper nouns, it is specific because it expresses information about one specific entity relative to another. Similarly, an assertion that contains a year is specific because it

contains specific temporal information.

2.3.3 Distinguishing Assertions

Another quality of interesting assertions might be providing distinguishing information about an object. This is related to novelty and surprisingness. Einstein may be a physicist who was born in Germany, but what really sets him apart and makes him interesting are his contributions to relativity theory and that he won the Nobel Prize. Conversely, assertions that do not set an object apart from other objects are often uninteresting.

We operationalize this notion of distinguishing using a technique similar to TF-IDF (term frequency - inverse document frequency) weighting [104]. In Information Retrieval, term frequency refers to the number of times a term occurs in a document. For our term frequency component, we define AssertionFrequency as the number of times an assertion occurs in the TextRunner set of assertions (*e.g.*, the number of times TextRunner found that "Einstein won the Nobel Prize"). For our document frequency component, we define ObjectFrequency as the number of times the object (*e.g.*, "the Nobel Prize") appears in a sample of ten million random TextRunner assertions. We define an AFOFRatio(Extraction) as follows:[2]

$$AFOFRatio(E) = \frac{AssertionFrequency(E)}{ObjectFrequency(object(E)) + 1} \tag{2.1}$$

For assertions, the AFOFRatio compares how often the assertion appears with how often we would expect the assertion to appear given its object. If the object has extremely high ObjectFrequency (*e.g.*, a common word like "food"), the AFOFRatio will be low. If the object has extremely low ObjectFrequency (*e.g.*, a misspelling or obscure term), then the AFOFRatio will be high. In the case of average ObjectFrequency, the AFOFRatio will reflect whether the assertion appears more often than one would normally expect.

Informal experimentation confirmed that extremely low AFOFRatio values often indicate an assertion is too vague to be interesting, while the very highest AFOFRatio values

[2]We add 1 in the denominator to prevent possible division by 0.

generally indicated assertions that were not well formed or well expressed. We chose a middle range ($1 < \text{AFOFRatio} \leq 10$) that seemed to generally yield interesting assertions from the distinguishing perspective.

2.3.4 Basic Assertions using Wikipedia

The final quality of interesting assertions that we focus on here are *basic* facts. These are definitional assertions that, for example, might be interesting to a person learning about a subject. A person learning about Einstein might look up such facts as "Einstein was a physicist" or "Einstein was born in 1879". Although it would be difficult to define all-encompassing rules for what makes an assertion basic, we can take advantage of the fact that knowledge bases such as Wikipedia sometimes provide high quality examples of basic knowledge. Many Wikipedia articles contain *infoboxes*, which are tabular summaries of basic information about objects. Our operationalization of basic assertions, is therefore based in learning a classifier to identify assertions similar to those that human editors have decided to include in infoboxes.

Training such a classifier requires examples of TextRunner assertions likely to reflect infobox knowledge (positive training examples) and assertions unlikely to reflect infobox knowledge (negative training examples). Starting with the DBPedia Wikipedia infobox database [2], we applied a series of filters and isolated a set of 872 entities with good infobox coverage. Text matching on infobox values (allowing for small edit distance) produced a set of 1,584 TextRunner assertions that reflected knowledge expressed in those infoboxes. This is comparable to how Kylin matches infobox data to statements [128], but our matching is stricter and thus achieves higher precision at lower recall. For the negative examples we then sampled 3,000 TextRunner assertions (about the same entities) that did not match any infobox values.

We train our basic classifier using around ten domain-independent features, such as the number of words in the assertion, whether the assertion relates proper nouns, whether the assertion ends on a stop word, and the estimated frequency of the assertion's object argument in TextRunner (refer to [65] for the full feature list). Lexical features (those

specific to the query terms, such as learning that any assertion with the relation "was born in" is interesting), are intentionally omitted because we are interested in a generally applicable classifier that is effective regardless of whether it was trained on assertions similar to those that it will classify (*e.g.*, "was born in" is useless on assertions about fruits). An experiment showed that the lexical features enable greater precision at the cost of reduced recall and generality.

One question to address here is that if we already have Wikipedia and we hypothesize that the infobox attributes are what is interesting, then why not just use all the Wikipedia data instead of involving Web extraction? Web extraction adds value beyond Wikipedia because Wikipedia is incomplete compared to the full Web. Many entities do not have Wikipedia articles, either because they have not been written yet or because they are not prominent enough to be in a general encyclopedia. The point is analyzed in further detail in Chapter 4. Even when an entity does have a Wikipedia article, often the infobox is incomplete or even missing. Also, while infobox attributes are good starting point for basic knowledge, there exist many additional similar attributes that also express basic knowledge but are not drawn out in infoboxes. Web extraction has the potential for much greater coverage, both in terms of entities covered and attributes per entity.

2.4 Evaluation

In order to evaluate our specific, basic, and distinguishing models, we used them each as the basis for filters that discard TextRunner results that fail to satisfy each model. To assess the quality of each filter, we conducted a study to collect human ratings of the interestingness of assertions.

2.4.1 Methods and Procedures

We first selected a set of ten study query terms including famous people (Albert Einstein, Bill Gates, Thomas Edison), other proper nouns (Beijing, Brazil, Microsoft, Diet Coke), improper nouns (sea lions), and also relationship queries (invented, destroyed). This query set is meant to provide a varied sample of the sorts of queries for which TextRunner can provide interesting results. Our analyses are based on the top thirty assertions resulting

from each of these queries, because the top results have the greatest impact on utility and about thirty results can be seen at a glance on a TextRunner results page.

As a baseline for comparison, we first obtain the number of times each assertion for each query was found by TextRunner. Our AssertionFrequency condition selects the thirty most frequently occurring assertions for each query. We next obtain assertions for our specific, distinguishing, and basic conditions by applying each of our filters in order of assertion frequency, discarding results that fail the filter, until we obtain thirty results that satisfy the filter. The study therefore focuses on 1200 assertions (10 queries * 4 conditions * 30 assertions).

We recruited 12 study participants (5 male, 7 female), who had a variety of backgrounds including math, marketing, finance, music, and nursing. Participants were each asked to rate 200 assertions on a scale from 1 (labeled "Least Interesting") to 5 (labeled "Most Interesting"). Assertions were presented one at a time, drawn randomly without replacement between participants. We gathered two to three ratings for every assertion, helping to account for individual differences in what people consider interesting.

2.4.2 Results

We analyze participant ratings of interestingness using a mixed model analysis of variance. We model our variable of interest, condition (values AssertionFrequency, simple, distinguishing, and basic), as a fixed effect. To account for learning or fatigue effects, we model trial number as a fixed effect. Similarly, we account for the possibility that how long a person viewed an assertion might impact their rating by modeling time to rate as a fixed effect. Finally, we account for variations in the interestingness of queries and variations in the ratings given by different people by modeling both query and participant as random effects.

We found no significant effect of either trial number or time to rate, and so remove both of them from the remainder of our analysis. The omnibus test reveals a significant main effect of condition ($F(4, 3542) = 15.6$, $p < .0001$), leading us to investigate pairwise differences. We use Tukey's Honestly Significant Difference (HSD) procedure to account

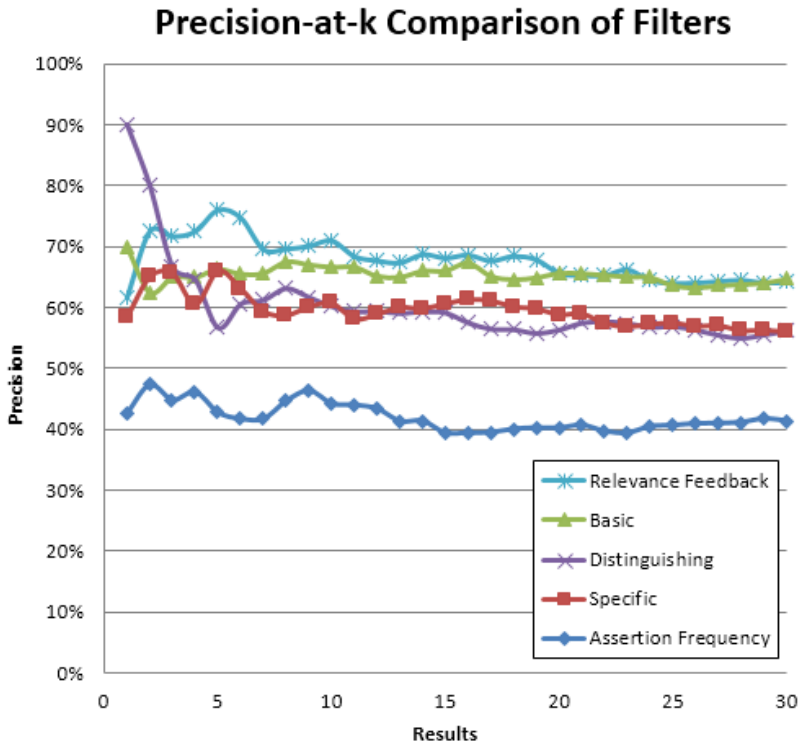

Figure 2.3: **Impact of Interestingness Filter:** Our trained filters led to significantly higher mean average precisions for whether top assertions were interesting. Relevance Feedback (67.9%) was the best ($p \approx .005$). Basic (65.4%) was second best ($p < .0001$). Specific (59.5%) and distinguishing (60.3%) were also better ($p < .0001$) than Assertion Frequency (the results without any filtering), which had the lowest mean average precision at 41.9%.

for increased Type I error in unplanned comparisons. This shows *basic* yielded the most interesting assertions, significantly more interesting than AssertionFrequency ($F(1,3545) = 55.0$, $p < .0001$), specific ($F(4,3539) = 7.7$, $p \approx .005$), and distinguishing ($F(1,3547) = 10.3$, $p \approx .001$). Our other filters also significantly improved interestingness, as both specific ($F(1,3544) = 21.2$, $p < .0001$) and distinguishing ($F(1,3539) = 18.7$, $p < .0001$) were significantly more interesting than AssertionFrequency.

2.4.3 Relevance Feedback

Although our results showed that basic assertions are the most interesting and that all of our filters yield results that are significantly more interesting than AssertionFrequency, inspec-

tion of our data suggested that our filters identify different interesting assertions. We found that only 21% of the interesting assertions would be identified by all three filters. We therefore consider whether a learning based method, using a classifier to combine information from all three filters, might perform better than any single filter.

In order to simplify this and our remaining analyses, we first reduce our five point scale to a binary classification. We define ratings of 4 or 5 to be interesting, define ratings of 1 or 2 to be not interesting, and ignore ratings of 3. This discretization creates a nearly even split of our collected human labels, which we then use as positive and negative training examples for a relevance feedback classifier.

We make the output of our specific, distinguishing, and basic filters available as features to this classifier. We also provide the same features that are used by the basic classifier. Because we are interested in a generally applicable classifier of interesting assertions, we evaluate trained relevance feedback classifiers using ten-fold cross validation such that we only test on the assertions from query terms not used to train the filter. This allows us to estimate performance on queries for which the system has not been trained, as we would expect improved performance on any queries for which the system has been trained.

Several classifiers from the WEKA toolkit [127] all had comparable precision-at-k values, averaging over ten-fold cross-validation, from k=1 to k=30. Decision Tree [95] was slightly better than the rest with an average precision at 67.9%. The precision at k measure illustrates the percentage of the first k results that are interesting, and is an appropriate and important measure because it corresponds to the quality of the top results.

2.4.4 Analysis

Figure 2.3 plots the precision-at-k for the relevance feedback decision tree classifier versus our specific, distinguishing, and basic filters as well as against AssertionFrequency. To test for difference between these curves, we conduct an analysis of variance for the precision at each plotted point, treating condition and k as fixed effects. The omnibus test reveals a significant main effect of condition ($F(4, 4) = 285$, $p < .0001$), leading us to investigate pairwise differences. We use Tukey's HSD procedure to account for increased Type I error

in unplanned comparisons. This shows that relevance feedback yields significantly more interesting assertions than specific ($F(1,144) = 95.4$, $p < .0001$), distinguishing ($F(1,144) = 78.6$, $p < .0001$), basic ($F(1,144) = 8$, $p \approx .005$) and AssertionFrequency ($F(1,144) = 926$, $p < .0001$).

The largest differences in Figure 2.3 are between our filter-based approaches and TextRunner's original use of AssertionFrequency, indicating the advantage of filtering. Appendix A further provides an example illustrating what the top 30 TextRunner extractions look like using our relevance feedback filter, compared to only sorting by frequency. The classifier filters trained with help from the knowledge base performed significantly better than all other approaches, indicating the utility of knowledge bases for this task. Our relevance feedback classifier achieves a precision at 30 of 64.1% and a mean average precision of 67.9%. This is comparable to human level performance, as we measured inter-annotator agreement in our label set to be approximately 70%.

2.4.5 Filtering in Practice

One motivation for this work was to create a filter that could increase the quality of results in the TextRunner question answering demo. We implemented a rule-based filter based on the ideas discussed in this chapter, adjusted slightly to run faster and trade some precision for increased recall. The filter runs when an argument or relation query is about to return so many results that it could benefit from filtering. The filter allows through extractions that are specific, while filtering extractions that are not distinguishing or that are so distinguishing that they are likely to be errors. Extractions could also be filtered using learned boundaries on what length extractions are most likely to be interesting (*e.g.*, extractions 30 to 79 characters in length are often more interesting). This filter worked well in practice at catching many of the uninteresting results, and filtered result sets were observably better. Versions of this filter were integrated into the TextRunner demo and demos for subsequent Open IE systems.

2.5 Conclusions

Extraction engines such as TextRunner are a promising avenue towards improving automated question answering and generating large knowledge bases. However, such systems are hamstrung by the fact that they often return uninformative results that are vague or uninteresting. Web extraction systems are particularly prone to this problem because of the general methods they use to extract entities and relationships [5]. This chapter has developed filters that allow TextRunner to better focus on assertions that are interesting. These filters aim to identify assertions that express specific information, distinguishing information, and basic information. We also designed an overall system that leverages all the filters we developed. This overall system is able to raise the average percentage of interesting results on a sample of queries significantly, from 41.6% to 64.1%.

Our strongest performing systems, the *basic* and *overall* systems, both leverage general knowledge base information in the form of the Wikipedia Infobox data used to help train the *basic* filter. By using only unlexicalized features that are not tied to any specific domains, we are able to learn classifiers for interestingness of general assertions. Domain independence is an important property to have when working with systems such as Open IE that emphasize the ability to operate over general Web text. To further validate this, our evaluation was conducted on a selection of query types across various domains. Identifying interesting Web extractions is an important problem, and we were able to solve it better by leveraging KBs.

The result of the research presented in this chapter is that we can now identify more interesting textual extractions from the Web. However, a central challenge remains that these extractions are still comprised only of text strings, and we lack any deeper understanding of their semantics. Instead of focusing on extractions such as ("Diet Coke", "is not", "a good idea") we can now focus on more meaningful extractions such as ("Diet Coke", "contains", "aspartame"). But what kind of relation does "contains" refer to? What kind of entity is "Diet Coke"? In the following chapters we will delve deeper into these types of questions, and especially into the role that large KBs can play in helping us address them.

Chapter 3

UNDERSTANDING RELATION PHRASES: THE FUNCTIONALITY PROPERTY

Web extractions from systems like Open IE contain textual relation phrases such as "was born in", "visited" and "is the symbol of". If two extractions share the same relation phrase then there is a good chance they express the same type of information. If two relation phrases tend to appear with the same argument pairs, then they may be synonymous. However we do not have any deeper understanding of the individual relation phrases beyond that.

One useful direction for further understanding of relation phrases it to identify which *relation properties* they each hold [66, 91]. For example, relations like "is smaller than" are *transitive*, meaning that $rel(a, b)$ and $rel(b, c)$ implies $rel(a, c)$. Relations like "is married to" are *symmetric*, meaning that $rel(a, b)$ implies $rel(b, a)$. Relations like "was born in" are *functional*, meaning that $rel(a, b)$ and $rel(a, c)$ implies that $b = c$. There exist many such relation properties, and learning which ones hold for which relations can give us deeper understanding of the nature of the relations and also enable numerous end-task applications.

Web text contains many more relation phrases than people could manually label with relation properties, so in this chapter we examine how relation properties could be *automatically* learned from large collections of Web extractions. As a focus, we specifically study the *functionality* property because functionality is useful for numerous tasks [68], and also because the technical challenges involved are shared by many other relation properties as well. We begin by exploring intuitive and statistical approaches to functionality identification. Keeping with the theme of this thesis, we then show how leveraging external knowledge bases can help with this task to enable even stronger performance. This chapter makes the following technical contributions:

1. We identify and enumerate the linguistic phenomena that make corpus-based identification of functionality and other relation properties surprisingly difficult.

2. We design and implement three novel techniques for identifying functionality based on instance-based counting, distributional differences, and the use of external knowledge bases.

3. Our final method for functionality detection outperforms the existing state-of-the-art significantly, increasing area under the precision-recall curve from 0.61 to 0.88.

3.1 The Functionality Property

The paradigm of Open Information Extraction (IE) [4, 5] has scaled extraction technology to the massive set of relations expressed in Web text. However, additional work is needed to better understand these relations, and to place them in richer semantic structures. A step in that direction is identifying the properties of these relations, *e.g.*, transitivity, symmetry, 1-to-1 [91], 1-to-many and our focus in this chapter – functionality. We refer to this problem as *functionality identification*.

A binary relation is functional if, for a given arg1, there is exactly one unique value for arg2. Examples of functional relations are *father, death_date, birth_city, etc.* We define a *relation phrase* to be functional if all semantic relations commonly expressed by that phrase are functional. For example, we say that the phrase *'was born in'* denotes a functional relation, because the different semantic relations expressed by the phrase (*e.g.*, *birth_city, birth_year, etc.*) are all functional.

Knowing that a relation is functional is helpful for numerous NLP inference tasks. Previous work has used functionality for the tasks of contradiction detection [100], quantifier scope disambiguation [117], and synonym resolution [130]. In Section 4.5.5 we describe how it could help with the important task of identifying ambiguous noun phrases. Functionality also has applications in other tasks such as ontology generation and information extraction. As an example application, consider two sentences from a contradiction detection task:

- "George Washington *was born in* Virginia."

- "George Washington *was born in* Texas."

As Ritter *et al.* [100] points out, in examples such as this, we can only determine that the two sentences are contradictory if we know that the semantic relation referred to by the phrase *'was born in'* is functional, and that Virginia and Texas are distinct states.

Automatic functionality identification is essential when dealing with a large number of relations as in Open IE, or in complex domains where expert help is scarce or expensive (*e.g.*, biomedical texts). This chapter tackles automatic functionality identification using Web text. While functionality identification has been utilized as a module in various NLP systems, previous work has not focused exclusively on functionality identification as a *bona fide* NLP inference task.

It is natural to identify functions based on triples extracted from text instead of analyzing sentences directly. Thus, as our input, we utilize tuples extracted by TextRunner [5] when run over a corpus of 500 million Web pages. TextRunner maps sentences to tuples of the form <arg1, *relation phrase*, arg2> and enables our system to focus on the problem of deciding whether the relation phrase is a function.

The naive approach, which classifies a relation phrase as non-functional if several arg1s have multiple arg2s in our extraction set, fails due to several reasons: synonymy – a unique entity may be referred by multiple strings, polysemy of both entities and relations – a unique string may refer to multiple entities/relations, metaphorical usage, extraction errors and more. These phenomena conspire to make the functionality determination task inherently statistical and surprisingly challenging.

In addition, a functional relation phrase may appear non-functional until we consider the types of its arguments. In our *'was born in'* example, <George Washington, *was born in*, 1732> does not contradict <George Washington, *was born in*, Virginia> even though we see two distinct arg2s for the same arg1. To solve functionality identification, we need to consider *typed relations* where the relations analyzed are constrained to have specific argument types.

We develop several approaches to overcome these challenges. Our first scheme employs approximate argument merging to overcome the synonymy and anaphora problems. Our second approach, DistrDiff, takes a statistical view of the problem and learns a separator for the typical count distributions of functional versus non-functional relations. Our third

scheme, CleanLists, identifies and processes a cleaner subset of the data by intersecting the corpus with entities in a general knowledge-base (in this case, Freebase [74]). Utilizing typing information, CleanLists first identifies typed functionality for suitable types for that relation phrase, and then combines them to output a final functionality label. Finally, we create a hybrid of CleanLists and DistrDiff, named Leibniz, that improves upon state-of-the-art performance for our task by combining both our statistical and knowledge base approaches.[1]

3.2 Related Work

There is a recent surge in large knowledge bases constructed by human collaboration such as Freebase [74] and VerbNet [57]. VerbNet annotates its verbs with several properties but not functionality. Freebase does annotate some relations with an 'is unique' property, which is similar to functionality, but the number of relations in Freebase is still much smaller than the hundreds of thousands of relations existing on the Web, necessitating automatic approaches to functionality identification.

Discovering functional dependencies has been recognized as an important database analysis technique [49, 129], but the database community does not address any of the linguistic phenomena which make this a challenging problem in NLP. Three groups of researchers have studied functionality identification in the context of natural language.

AuContraire [100]**:** is a contradiction detection system that also learns relation functionality. Their approach combines a probabilistic model based on [27] with estimates on whether each arg1 is ambiguous. The estimates are used to weight each arg1's contribution to an overall functionality score for each relation. Both argument-ambiguity and relation-functionality are jointly estimated using an EM-like method. While elegant, AuContraire requires substantial hand-engineered knowledge, which limits the scalability of their approach.

Lexico-syntactic patterns: Srinivasan and Yates [117] disambiguate a quantifier's scope by first making judgments about relation functionality. For functionality, they look for

[1] Leibniz is named after Gottfried Leibniz, the mathematician who coined the term "function".

numeric phrases following the relation. For example, the presence of the numeric term 'four' in the sentence "the fire *destroyed* four shops" suggests that *destroyed* is not functional, since the same arg1 can destroy multiple things.

The key problem with this approach is that it often assigns different functionality labels for the present tense and past tense phrases of the same semantic relation. For example, it will consider *'lived in'* to be non-functional, but *'lives in'* to be functional, since we rarely say "someone lives in many cities". Since both these phrases refer to the same semantic relation this approach has low precision. Moreover, it performs poorly for relation phrases that naturally expect numbers as the target argument (*e.g.*, *'has an atomic number of'*).

While these lexico-syntactic patterns do not perform as well for our task, they are well-suited for identifying whether a verb phrase can take multiple objects or not. This can be understood as a functionality property of the verb phrase within a sentence, as opposed to functionality of the semantic relation the phrase represents.

WIE: In a preliminary study, WIE [91] explored an instance based counting approach to various relation properties. WIE cannot automatically judge the functionality of thousands or millions of detected relation phrases. Instead, it requires humans to first manually annotate types on every relation to be judged. The majority of practical challenges for functionality detection that will be explored in this chapter (*e.g.*, how to judge relations without requiring type annotations, and how to judge even the relations that do not match any known types) are beyond the scope of the WIE study.

Finally, functionality is just one property of relations that can be learned from text. A number of other studies [24, 40, 122] have examined detecting other relation properties from text and applying them to tasks such as ontology cleaning.

3.3 Challenges for Functionality Identification

A functional binary relation r is formally defined as one such that $\forall x, y_1, y_2 : r(x, y_1) \wedge r(x, y_2) \Rightarrow y_1 = y_2$. We define a relation string to be functional if all semantic relations commonly expressed by the relation string are individually functional. Thus, under our definition, *'was born in'* and *'died in'* are functional, even though they can take different arg2s for the same arg1, *e.g.*, year, city, state, country, *etc.*

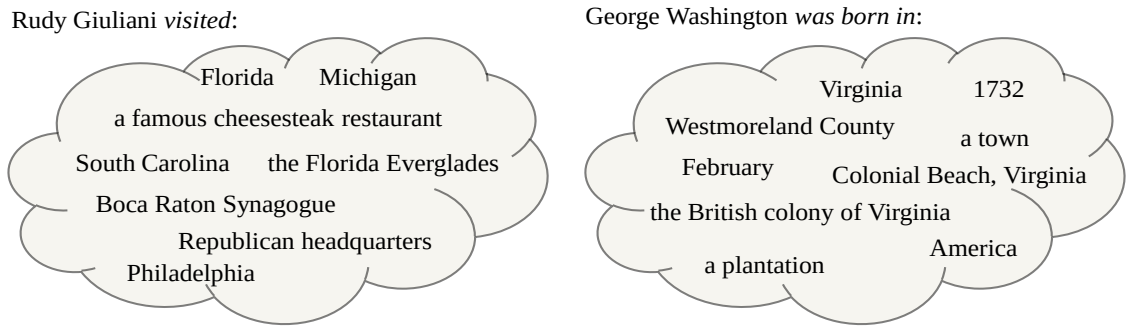

Figure 3.1: **Challenges in Functionality Detection:** Sample arg2 values for a non-functional relation (*visited*) *vs.* a functional relation (*was born in*) illustrate the challenge in discriminating functionality from Web text.

The definition of a functional relation suggests a naive instance-based counting algorithm for identifying functionality: "Look for the number of arg2s for each arg1. If all (or most) arg1s have exactly one arg2, label the relation phrase functional, else, non-functional." Unfortunately, this naive algorithm fails for our task exposing several linguistic phenomena that make our problem hard (see Figure 3.1):

Synonymy: Various arg2s for the same arg1 may refer to the same entity. This makes many functional relations seem non-functional. For instance, <George Washington, *was born in*, Virginia> and <George Washington, *was born in*, the British colony of Virginia> are not in conflict. Other examples of synonyms include 'Windy City' and 'Chicago', '3rd March' and '03/03', *etc.*

Anaphora: An entity can be referred to by using several phrases. For instance, <George Washington, *was born in*, a town> does not conflict with his being born in 'Colonial Beach, Virginia', since 'town' is an anaphora for his city of birth. Other examples include 'The US President' for 'George W. Bush', and 'the superpower' to refer to 'United States'. The effect is similar to that of synonyms – many relations incorrectly appear non-functional.

Argument Ambiguity: <George Washington, *was born in*, 'Kortrijk, Belgium'> in addition to his being born in 'Virginia' suggests that *'was born in'* is non-functional. However, the real cause is that 'George Washington' is ambiguous and refers to different people. This

ambiguity gets more pronounced if the person is referred to just by their first (or last name), *e.g.*, 'Clinton' is commonly used to refer to both Hillary and Bill Clinton (and less frequently other Clintons, such as George Clinton).

Relation Phrase Ambiguity: A relation phrase can have several senses. For instance '*weighs* 80 kilos' is a different *weighs* than '*weighs* his options'.

Type Restrictions: A closely related problem is type-variations in the argument. *E.g.*, <George Washington, *was born in*, America> *vs.* <George Washington, *born in*, Virginia> both use the same sense of *'was born in'* but refer to different semantic relations – one that takes a country in arg2, and the other that takes a state. Moreover, different argument types may result in different functionality labels. For example, *'published in'* is functional if the arg2 is a year, but non-functional if it is a language, since a book could be published in many languages. We refer to this finer notion of functionality as *typed functionality*.

Data Sparsity: There is often limited data for the more obscure relation phrases, and non-functional relation phrases may appear functional due to lack of evidence.

Textually Functional Relations: Last but not least, some relations that are not functional may appear functional in text. An example is *'collects'*. We collect many things, but rarely mention it in text. Usually, someone's collection is mentioned in text only when it makes the news. We say such relations are *textually functional*. Even though we could build techniques to reduce the impact of other phenomena, no instance based counting scheme could overcome the challenge posed by textually functional relations.

Finally, we note that our functionality predictor operates over tuples generated by an Open IE system. The extractors are not perfect and their errors can also complicate our analysis.

3.4 Algorithms

To overcome these challenges, we design three algorithms. Our first algorithm, IBC, applies several rules to determine whether two arg2s are equal. Our second algorithm, DistrDiff, takes a statistical approach, and tries to learn a discriminator between typical count distributions for functional and non-functional relations. Our final approach, CleanLists,

applies counting over a cleaner subset of the corpus, which is generated based on entities present in a large, general-coverage KB such as Freebase.

From this section onwards, we gloss over the distinction between a semantic relation and a relation phrase because our algorithms do not have access to relations and operate only at the phrase level. We use 'relation' to refer to the phrases.

*3.4.1 Instance Based Counting (*IBC*)*

For each relation, IBC computes a global functionality score by aggregating local functionality scores for each arg1. The local functionality for each arg1 computes the fraction of arg2 pairs that refer to the same entity. To operationalize this computation we need to identify which arg2s co-refer. Moreover, we also need to pick an aggregation strategy to combine local functionality scores.

Data Cleaning: Common nouns in arg1s are often anaphoras for other entities. For example, <the company, *was headquartered in*, ...> refers to different companies in different extractions. To combat this, IBC restricts arg1s to proper nouns. Secondly, to counter extraction errors and data bias, it retains an extraction only once per unique sentence. This reduces the disproportionately large frequencies of some assertions that are generated from a single article published at multiple websites. Similarly, it allows an extraction only once per website URL. Moreover, it filters out any arg1 that does not appear at least 10 times with that relation.

Equality Checking: This key component judges if two arg2s refer to the same entity. It first employs weak typing by disallowing equality checks that cross high-level categories (common nouns, proper nouns, dates and numbers). This mitigates relation ambiguity because, for instance, we never compare *'born in*(1732)' and *'born in*(Virginia)'. Within the same category it judges two arg2s to co-refer if they share a content word. It also performs a connected component analysis [47] to take a transitive closure of arg2s judged equal (see Figure 3.2). For example, for the relation *'was named after'* and arg1='Bluetooth', our corpus has three arg2s: 'Harald Bluetooth', 'Harald Bluetooth, the King of Denmark' and 'the King of Denmark'. Our equality method judges all three as referring to the same entity.

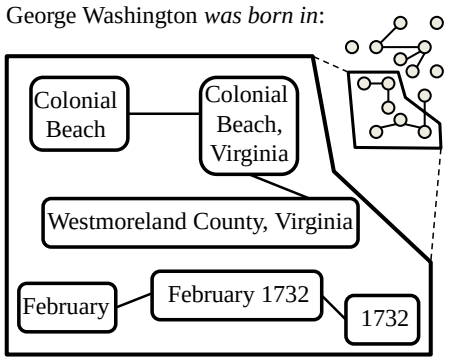

Figure 3.2: **Connected Components Technique:** Connected components analysis allows us to judge that *Colonial Beach* and *Westmoreland County, Virginia* are likely to be consistent arguments if we also see *Colonial Beach, Virginia*.

Note that this is a heuristic approach which could make mistakes. However, for an error to occur, there would need to be extractions with the same arg1, same relation, and similar arg2s. Such cases exist, but are not common. Our equality checking mitigates the problems of anaphora, synonymy as well as some typing.

Aggregation: We try several methods to aggregate local functionality scores for each arg1 into a global score for the relation. These include, a simple average, a weighted average weighted by frequency of each arg1, a weighted average weighted by log of frequency of each arg1, and a Bayesian approach that estimates the probability that a relation is functional using statistics over a small development set. Overall, the log-weighting works the best: it assigns a higher score for popular arguments, but not so high that it drowns out all the other evidence.

3.4.2 DistrDiff

Our second algorithm, DistrDiff, takes a purely statistical, discriminative view of the problem. It recognizes that, due to aforementioned reasons, whether a relation is functional or not, there are bound to be several arg1s that look locally functional and several that look locally non-functional. The difference is in the *number* of such arg1s – a functional relation will have more of the former type.

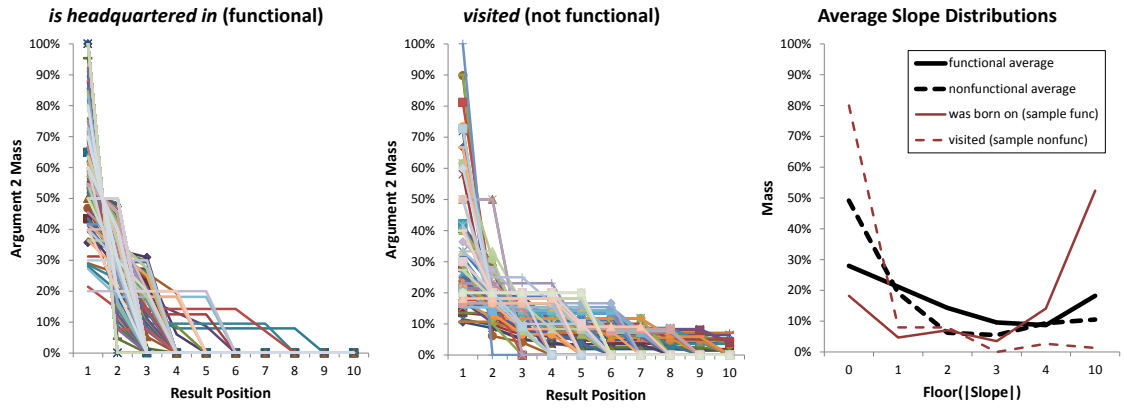

Figure 3.3: **Examining Argument Distributions:** Arg2 count distributions fall more sharply for (a) a sample functional relation, than (b) a sample non-functional relation. (c) The distance of aggregated slope-distributions from average slope-distributions can be used to predict the functionality.

DistrDiff studies the count distributions for a small development set of functional relations (and similarly for non-functional) and attempts to build a separator between the two. As an illustration, Figure 3.3(a) plots the arg2 counts for various arg1s for a functional relation (*'is headquartered in'*). Each curve represents a unique arg1. For an arg1, the x-axis represents the rank (based on frequency) of arg2s and y-axis represents the normalized frequency of the arg2. For example, if an arg1 is found with just one arg2, then x=1 will match with y=1 (the first point has all the mass) and x=2 will match with y=0. If, on the other hand, an arg1 is found with five arg2s, say, appearing ten times each, then the first five x-points will map to 0.2 and the sixth point will map to 0.

We illustrate the same plot for a non-functional relation (*'visited'*) in Figure 3.3(b). It is evident from the two figures that, as one would expect, curves for most arg1s die early in case of a functional relation, whereas the lower ranked arg2s are more densely populated in case of a non-functional relation.

We aggregate this information using *slope* of the best-fit line for each arg1 curve. For functional relations, the best-fit lines have steep slopes, whereas for non-functional the lines are flatter. We bucket the slopes in integer bins and count the fraction of arg1s appearing in each bin. This lets us aggregate the information into a single slope-distribution for each

relation. Bold lines in Figure 3.3(c) illustrate the average slope-distributions, averaged over ten sample relations of each kind – dashed for non-functional and solid for functional. Most non-functional relations have a much higher probability of arg1s with low magnitude slopes, whereas functional relations are the opposite. Notice that the aggregated curve for *'visited'* in the figure is closer to the average curve for non-functional than to functional and vice-versa for *'was born on'*.

We plot the aggregated slope-distributions for each relation and use the distance from average distributions as a means to predict the functionality. We use KL divergence [61] to compute the distance between two distributions. We score a relation's functionality in three ways using: (1) KLFUNC, its distance from average functional slope-distribution F_{avg}, (2) KLDIFF, its distance from average functional minus its distance from average non-functional N_{avg}, and (3) average of these two scores. For a relation with slope distribution R, the scores are computed as:

$$\text{KLFUNC} = \sum_i R(i) ln \frac{R(i)}{F_{avg}(i)} \tag{3.1}$$

$$\text{KLDIFF} = \text{KLFUNC} - (\sum_i R(i) ln \frac{R(i)}{N_{avg}(i)}) \tag{3.2}$$

Section 3.5.2 compares the three scoring functions. A purely statistical approach is resilient to noisy data, and does not need to explicitly account for the various issues we detailed earlier. A disadvantage is that it cannot handle relation ambiguity and type restrictions. Moreover, we may need to relearn the separator if applying DISTRDIFF to a corpus with very different count distributions.

3.4.3 CLEANLISTS

Our third algorithm, CLEANLISTS, is based on the intuition that for identifying functionality we need not reason over every instance for each relation; instead, a smaller but cleaner subset of the data may work best. This clean subset should ideally be free of synonyms, ambiguities and anaphora, and be typed.

Several knowledge-bases such as Wordnet [78], Wikipedia [126], and Freebase [74], are readily and freely available and they all provide clean typed lists of entities. In our experiments we chose to use Freebase because of its broad coverage of general entities that appear in Web text relation phrases. Freebase type lists also have the important property that they do not contain duplicates. For example, the list of *people* might include "Hillary Clinton" or "Hillary Rodham Clinton," but would not include both.

CleanLists takes the intersection of Freebase entities with our corpus to generate a clean subset for functionality analysis. Freebase contains over 10 million entities in over 1,000 typed lists. Thus, this intersection retains significant portions of the useful data while removing most anaphora and synonymy issues. Moreover, by matching against typed lists, many relation ambiguities are separated as well, since ambiguous relations often take different types in the arguments (*e.g.*, *'ran*(Distance)' *vs. 'ran*(Company)'). To mitigate the effect of argument ambiguity, we additionally remove instances in which arg1s match multiple names in the Freebase list of names.

As an example, consider the *'was born in'* relation. CleanLists will remove instances with only 'Clinton' in arg1, since it matches multiple people in Freebase. It will treat the different types, *e.g.*, cities, states, countries, months separately and analyze the functionality for each of these individually. By intersecting the relation data with argument lists for these types, we will be left with a smaller, but much cleaner, subset of relation data, one for each type. CleanLists analyzes each subset using simple, instance based counting and computes a typed functionality score for each type. Thus, it first computes typed functionality for each relation.

There are two subtleties in applying this algorithm. First, we need to identify the set of types to consider for each relation. Our algorithm currently picks the types that occur most in each relation's observed data. In the future, we could also incorporate a selectional preferences system [58, 101]. Note that we remove Freebase types such as *Written Work* from consideration for containing many entities whose primary senses are not that type. For example, both 'Al Gore' and 'William Clinton' are also names of books, but references in text to these are rarely a reference to the written work sense.

Secondly, an argument could belong to multiple Freebase lists. For example, 'California'

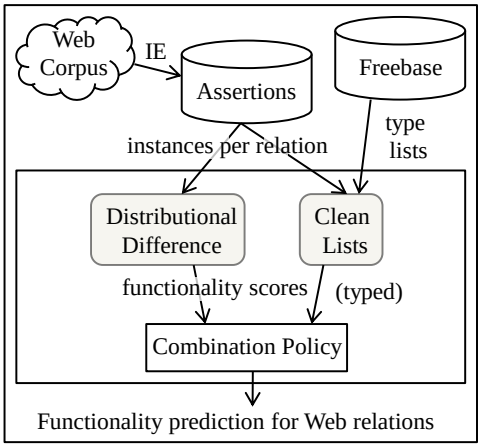

Figure 3.4: **Functionality Detection Components:** Our functionality system, LEIBNIZ, uses the Web and Freebase to determine functionality of Web relations.

is both a city and a state. We apply a simple heuristic: if a string appears in multiple lists under consideration, we assign it to the smallest of the lists (the list of cities is much larger than states). This simple heuristic usually assigns an argument to its intended type. On a development set, the error rate of this heuristic is <4%, though it varies a bit depending on the types involved. This technique also keeps us from potential errors caused by agglomerated types that contain multiple other types (*e.g.*, *location* type contains both *city* and *country* types).

CLEANLISTS determines the overall functionality of a relation string by aggregating the scores for each type. It outputs functional if a majority of typed senses for the relation are functional. For example, CLEANLISTS judges *'was born in'* to be functional, since all relevant type restrictions are individually typed functional – everyone is born in exactly one country, city, state, month, *etc.*

CLEANLISTS has a much higher precision due to the intersection with clean lists, though at some cost of recall. The reason for lower recall is that the approach has a bias towards types that are easy to enumerate. It does not have different distances (*e.g.*, 50 kms, 20 miles, *etc.*) in its lists. Moreover, arguments that do not correspond to a noun cannot be handled. For example, in the sentence, "He *weighed* eating a cheeseburger against eating a

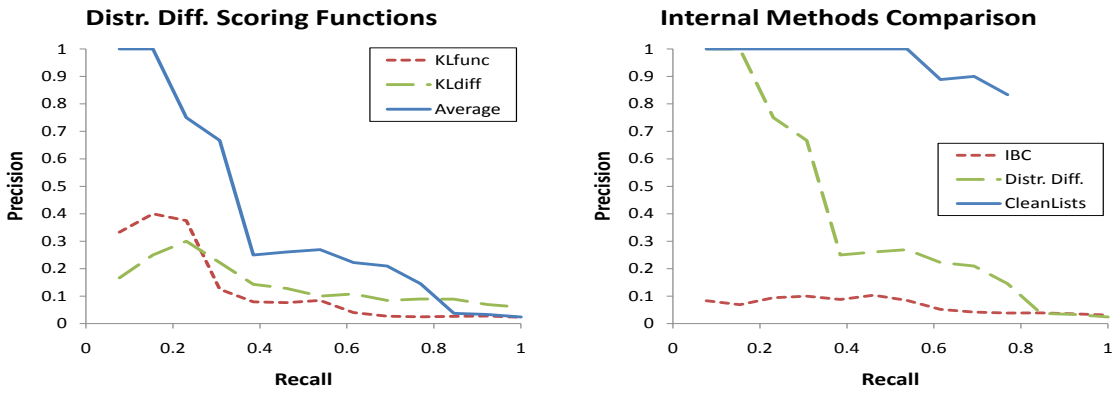

Figure 3.5: **Functionality Internal Comparisons:** (a) The best scoring method for DistrDiff averages KLfunc and KLdiff. (b) CleanLists performs significantly better than DistrDiff, which performs significantly better than IBC.

salad", the arg2 of *'weighed'* can't be matched to a Freebase list. To increase the recall we back off to DistrDiff in the cases when CleanLists is unable to make a prediction (see Figure 3.4). This combines the power of knowledge bases and statistical approaches to give the best balance of precision and recall for our task. We name our final system Leibniz.

One current limitation is that using only those arg2s that exactly match clean lists leaves out some good data (*e.g.*, a tuple with an arg2 of 'Univ. of Washington' will not match against a list of universities that spells it as 'University of Washington'). Because we have access to entity types, using typed equality checkers [92] with the clean lists would allow us to recapture much of this useful data. Argument matching will also be examined further in Chapter 4.

3.5 Evaluation

In our evaluation, we wish to answer three questions: (1) How do our three approaches, *Instance Based Counting* (IBC), DistrDiff, and CleanLists, compare on the functionality identification task? (2) How does our final system, Leibniz, compare against the existing state of the art techniques? (3) How well is Leibniz able to identify typed functionality for

different types in the same relation phrase?

3.5.1 Dataset

For our experiments we test on the set of 887 relations used by Ritter *et al.* [100] in their experiments. We use the Open IE corpus generated by running TextRunner on 500 million high quality Web pages [5] as the source of instance data for these relations. Extractor and corpus differences lead to some relations not occurring (or not occurring with sufficient frequency to properly analyze, *i.e.*, ≥ 5 arg1 with ≥ 10 evidence), leaving a dataset of 629 relations on which to test.

Two human experts tagged these relations for functionality. Tagging the functionality of relation phrases can be a bit subjective, as it requires the experts to imagine the various senses of a phrase and judge functionality over all those senses. The inter-annotator agreement between the experts was 95.5%. We limit ourselves to the subset of the data on which the two experts agreed (a subset of 601 relation phrases).

3.5.2 Internal Comparisons

First, we compare the three scoring functions for DistrDiff. We vary the score thresholds to generate the different points on the precision-recall curves for each of the three. Figure 3.5(a) plots these curves. It is evident that the hybrid scoring function, *i.e.*, one which is an average of KLfunc (distance from average functional) and KLdiff (distance from average functional minus distance from average non-functional) performs the best. We use this scoring in the further experiments involving DistrDiff.

Next, we compare our three proposed algorithms on the dataset. Figure 3.5(b) reports the results. CleanLists outperforms DistrDiff by vast margins, covering a 33.5% additional area under the precision-recall curve. Overall, CleanLists finds the very high precision points, because of its use of clean data. However, it is unable to make 23.1% of the predictions, primarily because the intersection between the corpus and Freebase entities results in very little data for those relations. DistrDiff performs better than IBC, due to its statistical nature, but the challenges described in Section 3.3 plague both these systems

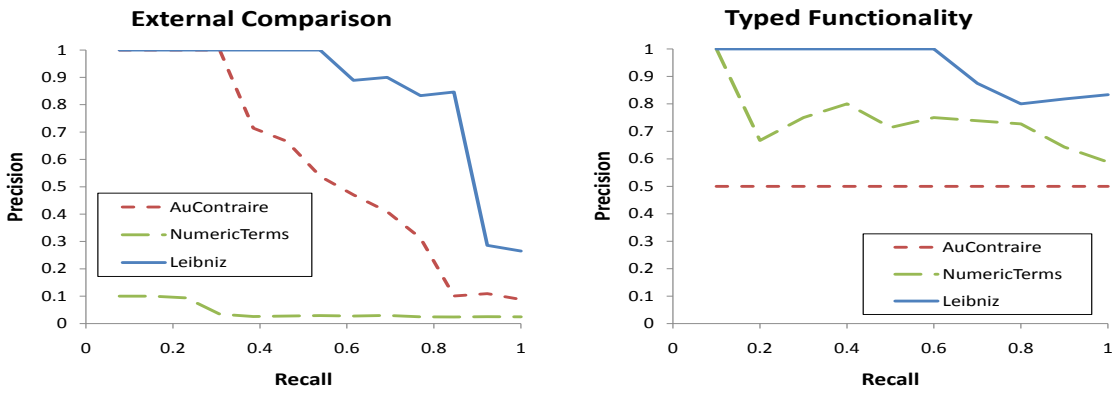

Figure 3.6: **Functionality External Comparisons:** (a) LEIBNIZ, which is a hybrid of CLEANLISTS and DISTRDIFF, achieves 0.88 AUC and outperforms the 0.61 AUC from AuContraire [100] and the 0.05 AUC from NumericTerms [117]. (b) LEIBNIZ is able to tease apart different senses of polysemous relations much better than other systems.

much more than CLEANLISTS.

3.5.3 External Comparisons

We next compare LEIBNIZ against the existing state of the art approaches. Our competitors are AuContraire [100] and NumericTerms [117]. Because we use the AuContraire dataset, we report the results from their best performing system. We reimplement a version of NumericTerms using their list of numeric quantifiers and extraction patterns that best correspond to our relation format. We run our implementation of NumericTerms on a dataset of 100 million English sentences from a crawl of high quality Web pages to generate the functionality labels.

Figure 3.6(a) reports the results of this experiment. We find that LEIBNIZ outperforms AuContraire by vast margins covering an additional 44% area in the precision-recall curve. AuContraire's AUC is 0.61 whereas LEIBNIZ covers 0.88. A Bootstrap Percentile Test [54] on F_1 score found the improvement of our techniques over AuContraire to be statistically significant at $\alpha = 0.05$. NumericTerms does not perform well, because it makes decisions based only on the local evidence in a sentence, and does not integrate the knowledge from

different occurrences of the same relation. It returns many false positives, such as *'lives in'*, which appear functional to the lexico-syntactic pattern, but are clearly non-functional, *e.g.*, one could live in many places over a lifetime.

An example of a LEIBNIZ error is the *'represented'* relation. LEIBNIZ classifies this as functional, because it finds several strongly functional senses (*e.g.*, when a person represents a country), but the human experts might have had some non-functional senses in mind while labeling.

3.5.4 Typed Functionality

Next, we conduct a study of the typed functionality task. We test on ten common polysemous relations, each having both a functional and a non-functional sense. An example is the *'was published in'* relation. If arg2 is a year it is functional, *e.g.* <Harry Potter 5, *was published in*, 2003>. However, *'was published in*(Language)' is not functional, *e.g.* <Harry Potter 5, *was published in*, [French / Spanish / English]>. Similarly, *'will become*(Company)' is functional because when a company is renamed, it transitions away from the old name exactly once, *e.g.* <Cingular, *will become*, AT&T Wireless>. However, *'will become*(government title)' is not functional, because people can hold different offices in their life, *e.g.*, <Obama, *will become*, [Senator / President]>.

In this experiment, a simple baseline of predicting the same label for the two types of each relation achieves a precision of 0.5. Figure 3.6(b) presents the results of this study. AuContraire achieves a flat 0.5 because it cannot distinguish between types. NumericTerms can be modified to distinguish between basic types – check the word just after the numeric term to see whether it matches the type name. For example, the modified NumericTerms will search the Web for instances of *"was published in* [numeric term] *years" vs. "was published in* [numeric term] *languages"*. This scheme works better when the type name is simple (*e.g.*, "languages") rather than complex (*e.g.*, "government titles").

LEIBNIZ performs the best and is able to tease apart the functionality of various types very well. When LEIBNIZ did not work, it was generally because of textual functionality, which is a larger issue for typed functionality than general functionality. Of course, these re-

sults are merely suggestive – we perform a larger-scale experiment and generate a repository of typed functions next.

3.5.5 Creating a Repository of Functional Relations

We now report on a repository of typed functional relations generated automatically by applying LEIBNIZ to a large collection of relation phrases. Instead of starting with the most frequent relations from TEXTRUNNER, we use REVERB's relations [37] because they are more specific. For instance, where TEXTRUNNER outputs an underspecified tuple, <Gold, *has*, an atomic number of 79>, REVERB extracts <Gold, *has an atomic number of*, 79>. REVERB enables LEIBNIZ to identify far more functional relations than TEXTRUNNER.

Scaling up to a large collection of typed relations requires us to consider the size of our data sets. For example, consider which relation is more likely to be functional—a relation with 10 instances all of which indicate functionality versus a relation with 100 instances where 95 behave functionally.

To address this problem, we adapt the likelihood ratio approach from Schoenmackers *et al.* [107]. For a typed relation with n instances, f of which indicate functionality, the G-test [29] provides a measure for the likelihood that the relation is not functional:

$$G_{score} = 2 * (f * ln(\frac{f}{k}) + (n - f) * ln(\frac{n - f}{n - k})) \tag{3.3}$$

Here k denotes the evidence indicating functionality for the case where the relation is not functional. Setting $k = 0.25$ * n worked well for us. This G-score replaces our previous metric for scoring functional relations.

Typing increases the number of relations significantly, and the exact set of type lists affects the quality of the results. Lists that contain ambiguous names can lead to noisy results for specific typed relations. For example, Freebase's list of *films* contains 73,000 entries, many of which (*e.g.*, "*Egg*") are not films in their primary senses. Even with heuristics such as assigning terms to their smallest lists and disqualifying dictionary words that occur from large type lists, there is still significant noise left. We chose a set of 35 clean lists that would work better for this task. They are nearly all from Freebase, except a few

(*e.g.*, *years*) that we could automatically generate at perfect accuracy. Using these clean lists on ReVerb's extraction corpus, we generated a repository of 5,520 typed functional relations.

To evaluate this resource a human expert tagged a random subset of the top 1,000 relations. Of these relations 22% were either ill-formed or had non-sensical type constraints. From the well-formed typed relations the precision was estimated to be 0.8. About half the errors were due to textual functionality and the rest were Leibniz errors. An example of a textually functional relation found is *wasTheFounderOf(company)*. Some examples of good functions found include *isTheSequelTo(videogame)* and *areTheBirthstoneFor(month*). Appendix B provides 15 examples of good functional relations identified here.

This is the first public repository of automatically-identified functional relations. Scaling up our data set forced us to confront new sources of noise including extractor errors, errors due to mismatched types, and errors due to sparse evidence. Still, our initial results are encouraging and we hope that our resource will be valuable as a baseline for future work.

3.6 Conclusions

Functionality identification is an important subtask for Web-scale information extraction and other machine reading tasks. We study the problem of predicting the functionality of a relation phrase automatically from Web text. We presented three algorithms for this task: (1) instance-based counting, (2) DistrDiff, which takes a statistical approach and discriminatively classifies the relations using average arg-distributions, and (3) CleanLists, which performs instance based counting on a subset of clean data generated by intersection of the corpus with a large, general knowledge base like Freebase.

Our best approach, Leibniz, is a hybrid of DistrDiff and CleanLists, and outperforms the existing state-of-the-art approaches by covering 44% more area under the precision-recall curve. We also observe that an important sub-component of identifying a functional relation phrase is identifying typed functionality, *i.e.*, functionality when the arguments of the relation phrase are type-constrained. Because CleanLists is able to use typed lists, it can successfully identify typed functionality. We run our techniques on a large set of relations to output a first repository of typed functional relations. We release this

data for further use by the research community.[2]

Two areas of future work are to better handle the textually functional relations and the relations with very few instances. If a non-functional relation appears functional in text, then this affects all approaches which detect properties via observed instances. One way to address this would be to carefully integrate evidence from pattern-based approaches that are more resilient to this problem. For example, if a relation is ever followed in text by "multiple" or "more than one," then it is less likely to be functional. While the G-test approach discussed in Section 3.5.5 helps us interpret results in cases of data sparsity, it would be even better to acquire more instances per relation by clustering any synonymous relation phrases.

We would also like to adapt our functionality techniques to detect other important relation properties. The primary technical challenges in detecting other properties often center around the same phenomena (*e.g.*, synonymy, anaphora, argument ambiguity, relation ambiguity). Several properties such as 1-to-1 and temporal functionality are direct extensions on functionality, so all the work here is directly applicable. Techniques we developed for IBC can help with properties where argument equality is important (*e.g.*, symmetry). DistrDiff can help with detecting the properties where specific argument distributions are expected (*e.g.*, 1-to-many). CleanLists can help identify the clean subsets of observed data to match on for properties such as transitivity, where the presence of any ambiguous arguments or anaphoric expressions can lead to false positives.

Having examined the problem of improving open-domain Web extractions as a whole (*e.g.*, "Einstein was born in 1879" is *interesting*) and then having delved deeper into understanding their relation phrases (*e.g.*, "was born in" is *functional*, so Einstein could not have been born in any other year), in the next chapter we will explore what additional information can be learned about the *extraction arguments*. For example, does "Einstein" refer to "Albert Einstein" or another Einstein? Does "Einstein" refer to a *person* or a *movie*? How might we determine this if a priori we knew nothing about Einstein, as is the case for many entities found on the Web?

---

[2]available at `http://www.cs.washington.edu/research/leibniz`

Chapter 4

UNDERSTANDING ENTITY PHRASES: LINKING AND TYPING

Web extractions from Open IE capture relationships between *entity phrases*. For example, ("Obama", "was born in", "Hawaii") implies that there exists entities "Obama" and "Hawaii" taking part in a "was born in" relationship. These entity phrases are extracted as text strings and have no additional associated semantic information. In this chapter we explore how these entity phrases can be linked into large knowledge bases, and how they can be marked up with semantic information such as *entity types*. Doing this would provide us with a deeper understanding of the entities within our extractions, facilitate interoperability between Web extractions and external projects/resources such as those found in Linked Data [9], and also enable the usage of Open IE extractions for numerous tasks such as typed question answering [16], inference [106], and more.

We begin the chapter by proposing scalable techniques for "entity linking" [15, 23] large corpora of Web extractions into Wikipedia. Successful linking for the earlier example would identify that "Obama" refers to "Barack Obama" rather than "Michelle Obama" or any other Obama. All Wikipedia entities have been annotated with Freebase semantic types, so this linking would further provide the types for all linked entities. In keeping with our goal of leveraging KBs without restricting the possible vocabulary of extracted knowledge, we then study how to also incorporate all the Web extraction arguments that do not have Wikipedia entries. We propose a pipeline of first detecting whether they represent entities (as opposed to errors), and then further predicting their types by identifying similar entities that we could link. The primary contributions of this chapter are:

1. We are the first to study how to efficiently entity link collections of millions of extractions. We introduce new techniques such as *collective contexts* and *inlink ratio* that are designed to operate at this scale. An implementation of these methods was able to entity link 500 million Web extractions in 3 days.

2. We present a method for discriminating *entities* from arbitrary noun phrases by utilizing features derived from temporal corpora [76]. Our system achieves 24% higher accuracy relative to a Named Entity Recognition baseline for this task.

3. We adapt and scale instance-to-instance class propagation [59] in a novel way in order to associate types with non-Wikipedia entities. On the task of assigning fine-grained types, we achieve twice the precision of our best baseline method.

4. After identifying a potential limitation around *ambiguous* noun phrases, we propose a method for splitting ambiguous noun phrases into their component senses by using graph coloring [52].

4.1 Entity Linking at Web Scale

In this section we provide an overview of entity linking and how we entity link millions of Web extractions.

4.1.1 Introduction

Given text, the task of *entity linking* [15, 23, 60, 79] is to identify knowledge base entities within the text, and mark each with its corresponding knowledge base entry. Nearly all general purpose linking systems use Wikipedia as the knowledge base because of its broad general coverage and to leverage its article texts and link structure during the linking process. In domain-specific contexts however, other entity catalogs can be used instead. An example of entity linking to Wikipedia is linking "New York" in the assertion ("New York", "acquired", "Pineda") to the Wikipedia article entry for *New York Yankees* (rather than, for example, the entries for *New York City* or *New York State*).

Entity linking elevates extraction arguments from text strings into meaningful *entities* that are disambiguated and have semantic types and linked resources. For example, linking to Wikipedia provides *Freebase types* for all linked entities.[1] Freebase contains over 1,000 semantic types and was designed to have good coverage across all entities within Freebase

[1]Wikipedia to Freebase mappings are available at http://download.freebase.com/wex

Extraction	Linked Entity (Freebase ID)	Type sample
("Obama", "was born in", "Hawaii")	Barack Obama (02mjmr)	*president*
("Obama", "attended", "Princeton")	Michelle Obama (025s5v9)	*celebrity*
("Einstein", "was born in", "Ulm")	Albert Einstein (0jcx)	*academic*
("Kenya", "is located in", "eastern Africa")	Kenya (019rg5)	*country*
("the Kiwi", "is a symbol of", "New Zealand")	Kiwi (04f85)	*organism*
("the FDA", "approved", "Sucralose")	Food & Drug Admin (032mx)	*gov agency*
("Titanic", "sank in", "the North Atlantic")	RMS Titanic (06l72)	*ship*

Table 4.1: **Entity Linking:** Entity Linking involves linking textual arguments to their disambiguated and typed entries in a large knowledge base.

and Wikipedia, and thus good coverage across the general space of Web entities. Freebase types are at a more suitable granularity for many end tasks than Wikipedia's own category system, which has over 100,000 categories (including many such as *1872 births* and *1983 horror films* that are too fine-grained to be practically useful for general reasoning tasks). Table 4.1 provides several examples of Web extractions, the entity they would link to, and sample types from each entity's actual list of Freebase types. Note that most entities have multiple Freebase types. Barack Obama's full list of types includes *president*, *author*, *person*, *public speaker* and many more.

Our goal is to entity link our Web-scale extraction set. The scale of the problem introduces numerous opportunities and challenges. Existing entity linking research has focused primarily on linking all the entities within *individual documents* into Wikipedia [28, 60, 79]. To link a million documents they would repeat a million times. However, we can potentially do better when we know ahead of time that the task is large scale linking. For example, information on one document might help link an entity on another document. Another opportunity is that after linking a million documents, we can discover systematic linking errors when particular entities are linked to many more times than expected.

Speed is a practical concern when linking this many assertions. Instead of designing a system with sophisticated features that rely on the full Wikipedia graph structure, the core

of our linker is fast linking features such as string matching, prominence [36], and context matching [15]. Ratinov *et al.* [98] found that these features already provide a baseline that is very difficult to beat with the more sophisticated features that take more time to compute. We also increase precision by incorporating new *corpus-level features* available only when linking large sets of extractions.

This section investigates entity linking over millions of high-precision extractions from a corpus of 500 million Web documents, toward the goal of creating a useful knowledge base of general facts. This work is the first to report on entity linking over this many extractions [69].

4.1.2 Entity Linking

We define our problem of corpus-level linking of Web extractions as: Given an entity catalog W containing n entities e_1, e_2, ..., e_n, and a corpus R of m unique textual assertions a_1, a_2, ..., a_m, for each assertion $a \in R$ find entity $e \in W$ that corresponds to each entity argument of a. The sizes n and m are very large. We focus initially on linking the subject argument for each a, but the techniques used are also applicable to linking object arguments.

For W we use Wikipedia, which covers over 3 million entities e. Each Wikipedia entity comes associated with a Freebase entity and we use Freebase types later, so in many of our examples we will refer to Wikipedia entities via their unique Freebase IDs. For R we initially use 15 million ReVerb extractions publicly released by Fader *et al.* [37].[2]

Our general algorithm has 3 components:

1. Candidate generation: Every Wikipedia entity $e \in W$ contains a Wikipedia page. The number of hyperlinks in all of Wikipedia that point to the page for e is the inlink count $inlinks(e)$. This serves as a prominence prior because prominent entities have more pages linking to them in Wikipedia [36]. At a high level, our candidate generation algorithm is:

List W_{sorted} = sort W by inlinks, from most to fewest
for each entity $e \in W_{sorted}$
  for each word $w \in e$

[2]available at http://reverb.cs.washington.edu

add e to the end of $words\_to\_candidates(w)$

for each assertion $a \in R$

add the subject argument of a to a list L

for each subject argument $s \in L$

$candidates_s$ = first k entities e where for every word $w \in s$, $e \in words\_to\_candidates(w)$

return $candidates$

Setting $k = 5$, this quickly gives us the top 5 most prominent candidates $e \in W$ that contain at least every word present in each textual argument. For example, for the argument "Clinton," this returns: *Bill Clinton*, *Hillary Clinton*, *George Clinton*, *Clinton County*, and *Presidency of Bill Clinton*. The most prominent exact string match in the candidates list is also included if it did not otherwise qualify. We use simple heuristics to handle differences in capitalization, pluralization, punctuation, and presence of stopwords. We found that broadly incorporating looser string matching, acronym matching and large alias lists often presented tradeoffs between precision and recall [83]. While we continue to explore their use, our initial candidate generation uses only the described matching.

2. Candidate ranking: For each assertion $a \in R$, we create a context document $context_a$ that contains all source sentences from which a was extracted. $article_e$ refers to the text of the Wikipedia article for entity e. We then compute a *link score* as follows:

for each assertion $a \in R$ with subject argument s

for each candidate entity match e in $candidates_s$

$sim(a, e) = cosine\ similarity(context_a, article_e)$

$sml(a, e)$ = string match level(s, e)

$link\_score(a, e)$ = ln(inlinks(e)) * $sim(a, e)$ * $sml(a, e)$

Given weighted query vectors V($context_a$) and V($article_e$):

$$cosine\ similarity(context_a, article_e) = \frac{V(context_a) \cdot V(article_e)}{||V(context_a)||\ ||V(article_e)||} \tag{4.1}$$

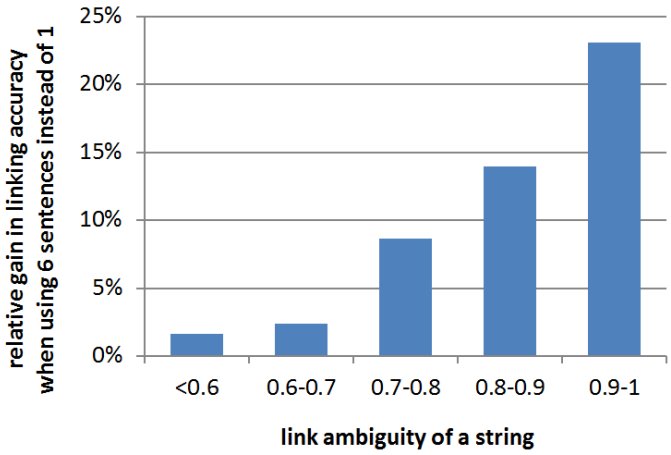

Figure 4.1: **Collective Contexts:** Context matching using more source sentences can increase entity linking accuracy, especially in cases where link ambiguity is high.

We refer to this usage of context documents as *collective contexts*, and it is a corpus-level feature that is only available when linking many assertions at once. The idea behind it is that in an extraction corpus, the same assertion is often extracted from multiple source sentences across different documents. If we collect together the various source sentences, this can often provide a stronger context signal for entity linking. While "New York acquired Pineda on January 23" may not provide strong signal by itself, adding another source sentence such as "New York acquired Pineda to strengthen their pitching staff," could be enough for the linker to choose the *New York Yankees* over the others. Figure 4.1 shows the gain in linking accuracy we observed when using 6 randomly sampled source sentences per assertion instead of 1. At each *link ambiguity* level (Equation 4.3), we took 200 random samples.

String match level is an integer from 1 to 5 that corresponds to how closely the strings match (5 = exact, 1 = e has many more words than s). We take the logarithm of the inlink count to keep this value from dominating the others. The best entity link ê for assertion a with subject argument s maximizes the link score:

$$\hat{e}(a) = \{e | link\_score(a, e) = \max_{e \in candidates_s} link\_score(a, e)\} \tag{4.2}$$

entity	assertions	wiki inlinks	ratio
"Barack Obama"	16,094	16,415	0.98
"Larry Page"	13,871	588	23.6
"Bill Clinton"	5,710	11,176	0.51
"Microsoft"	5,681	12,880	0.44
"Same"	6,975	36	193

Table 4.2: **Inlink Ratio:** The ratio between an entity's *linked assertion count* and its *inlink prominence* can help to detect systematic errors to correct or filter out.

When there are multiple candidates, we use the score é(a) of the next best match to calculate a *link ambiguity* score:

$$link\_ambiguity(\hat{e}(a)) = \frac{link\_score(\acute{e}(a))}{link\_score(\hat{e}(a))} \tag{4.3}$$

Link ambiguity corresponds to whether the best candidate is clearly better than all other candidates. Links with high link score and low link ambiguity are more likely to be correct.

3. Filtering: A second corpus-level feature we found to be useful is *link count expectation*. When linking millions of general assertions, we do not expect strong relative deviation between the number of assertions linking to each entity and the *known prominence of the entities*. For example, we would expect many more Web assertions to link to "Lady Gaga" than "Michael Pineda." We formalize this notion by calculating an *inlink ratio* for each entity as the number of assertions linking to it divided by its inlink prominence.

$$inlink\ ratio(e) = \frac{\sum_{a \in R}(1 \text{ if } e = \hat{e}(a),\ 0 \text{ otherwise})}{inlinks(e)} \tag{4.4}$$

When linking 15 million assertions, we found that ratios significantly greater than 1 were often signs of systematic errors. Table 4.2 shows ratios for several entities that had many

assertions linked to them. It turns out that many assertions of the form "(Page, loaded in, 0.23 seconds)" were being incorrectly linked to "Larry Page," and assertions like "(Same, goes for, women)" were being linked to a city in East Timor named "Same." We filtered systematic errors detected in this way, but these errors could also serve as valuable negative labels in training a better linker.

4.1.3 Speed

Some existing linking systems we looked at (such as [44, 98]) can take up to several seconds to link each document, which makes them difficult to run at Web scale without massive distributed computing. By focusing on the fastest local features and then improving precision using corpus-level features, our initial implementation was able to link at an average speed of 60 assertions per second on a standard machine without using multithreading. This translated to 3 days to link the set of 15 million textual assertions that ReVerb identified as having high precision. A distributed implementation of this linker running on 9 multi-core machines was able to link 500 million ReVerb assertions in 3 days.

4.1.4 Accuracy

Of the 15 million assertions we tried to link, approximately 5 million could not be linked at all (generally because they had no string match). These 5 million were primarily non-Wikipedia entities and errors, although about 10% were also aliases that we would need to engineer in good alias lists [116] to link. We revisit these 5 million in Section 4.3. Of the 10 million that could be linked, we filter out 3 million for bad inlink ratios ($\geq$1). Bad inlink ratios generally indicated errors, such as "(Turn, left on, Erwin Road)" being linked to an Irish band named "Turn." From the 7 million assertions that we linked and which had good inlink ratios, we randomly sampled 100 and hand-labeled each as having a correct link, an incorrect link, not being an entity, or being an entity missing from our KB. We then used these labels to extrapolate the linking accuracy over the full set of 7 million assertions.

We found that our links were generally correct when link score was high and link ambiguity was low, and generally incorrect when link score was low and link ambiguity was high.

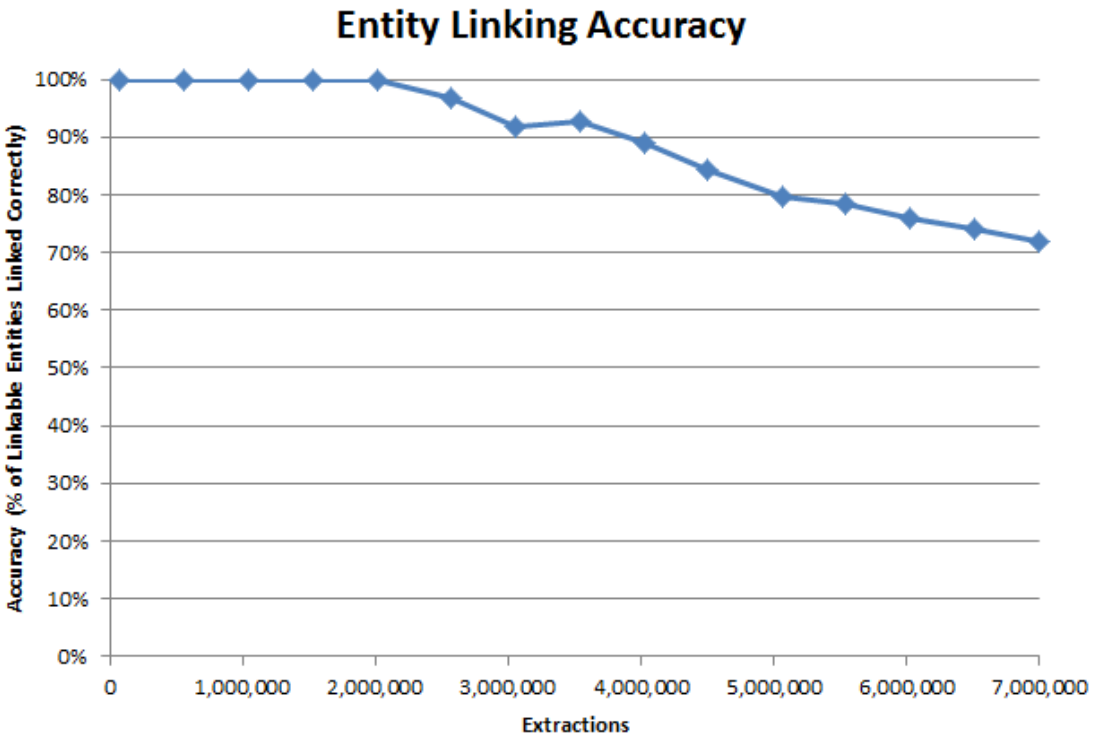

Figure 4.2: **Entity Linking Accuracy:** There are 2 million extractions where we correctly link nearly all the linkable entities, and 4 million extractions where we correctly link nearly 90% of the linkable entities.

After sorting the links by *link score* x (1 - *link ambiguity*), we created Figure 4.2 showing the linking accuracy at different numbers of assertions. Among the first 2 million assertions (based on our sort), we correctly linked nearly all of the entities that had correct links to find. Among the first 4 million assertions, we correctly linked around 90% of the linkable entities. Over the full set of 7 million assertions, we correctly linked just over 70% of the linkable entities. This evaluation confirms that we are able to accurately link millions of assertions for use in downstream applications.

Incorrect links were due to a variety of different issues. Around 1/3 of incorrect links occurred when the argument was a very common name (*e.g.*, "Max" or "Lacy") that matched a large number of potential entities. Another 1/3 of incorrect links were due to our current string matching logic. For example, when linking "HOL," our string matching logic

favors entities such as "HOL theorem prover" instead of catching that "HOL" can be an abbreviation of "Higher-order logic." Other sources of error included strongly correlated entities (*e.g.*, linking "The Aptera" to "Aptera Motors" instead of the "Aptera 2 Series" of cars made by Aptera Motors) and cases where the source sentences did not provide enough distinguishing context to overcome the prominence priors.

The accuracy metric for Figure 4.2 does not consider extractions that had no correct link to find. In our labeled test set, around one third of the extractions had no correct link to find (mostly because they referred to entities outside of our KB). The traditional way to handle these cases would be to assign them to a separate NIL label [28, 89]. The large majority of these cases occurred when link score was low and link ambiguity was high. If we assign the bottom half of our links to be NIL based on *link score* x (1 - *link ambiguity*), then accuracy (redefined as "% Correct, including NIL cases") becomes 65% over the full set of 7 million. This value could be further improved by learning the optimal link score and link ambiguity thresholds for triggering a NIL prediction.

Another question is how much precision and recall we lose in order to develop a system fast enough to link at Web scale. As computing power continues to increase in the future, will it eventually become a better option to preprocess Web pages with traditional entity linking? To test this we sampled 100 random extractions where the subject argument links into Wikipedia, and compared our linking ability against that of a strong off-the-shelf entity linker [98] which is given source Web pages to link. On our test set, the off-the-shelf linker assigned 78 links and got 72 correct, to achieve 92% precision at 72% recall (0.808 F_1). In comparison, our system was faster but had lower F_1. At a link score threshold of 5, we assigned 71 links and got 61 correct, to achieve 86% precision at 61% recall (0.714 F_1). At a link score threshold of 11, we assigned 51 links and got 50 correct, to achieve 98% precision at 50% recall. The off-the-shelf linker excels at linking individual extractions, but at Web scale our linker can produce a larger set of correctly linked extractions by setting a high link score threshold and processing many more Web pages in the same amount of time.

Appendix C contains the most-linked-to entities from our run on 15m Web extractions. The top entities (*e.g.*, Obama, Jesus, Internet, USA) are consistent with what entities we would expect to be mentioned most in Web text.

4.1.5 Conclusions

While numerous entity-linking systems have been developed in recent years, we believe that going forward, researchers will increasingly be considering the opportunities and challenges that arise when assigning millions of links at once, instead of linking document by document. This work is the first to run and report on entity linking over millions of textual extractions, and we proposed novel ideas in areas such as corpus-level features.

There are potentially many other corpus-level features and characteristics to explore, as well as additional challenges such as how to best evaluate recall at this scale and how to incorporate the many entities in Web text are not present in large knowledge bases. Of the 15m extractions we started with, around 5m extractions had no matches (mostly due to no close string matches). There were 1.4m distinct noun phrase subject arguments among these 5 million extractions. Noun phrases referring to entities not in the KB are *unlinkable* because there is no correct entry to link them to, and we study them next.

4.2 The Unlinkable Noun Phrase Problem

A problem we observed is that despite containing 3 to 4 million entities, Wikipedia does not cover a significant number of entities found in Web extractions. This occurs with entities that are not prominent enough to have their own dedicated article and with entities that are very new. Consider the sentence "Some people think that pineapple juice is good for vitamin C." To analyze this sentence, a machine should know that "pineapple juice" refers to a beverage, while "vitamin C" refers to a nutrient. Because it links into a fixed knowledge base, entity linking has a limited and somewhat arbitrary range. In our example, linkers (including those by [38] and [98]) generally link "vitamin C" correctly, but link "pineapple juice" to "pineapple." "Pineapple juice" is not entity linked as a beverage because it is not prominent enough to have its own Wikipedia entry. As Table 4.3 shows, Wikipedia often has prominent entities, while missing tail and new entities of the same types.[3] Wang *et al.* [123] notes that there are more than 900 different active shoe brands, but only 82 exist in Wikipedia. Facebook has over 800 million users, and each of them could be considered

[3]The same problem occurs with Freebase, which is also missing the same Table 4.3 entities.

example noun phrases	Wikipedia status
"apple juice" "orange juice"	present
"prune juice" "wheatgrass juice"	**absent**
"radiation exposure" "workplace stress"	present
"asbestos exposure" "financial stress"	**absent**
"IJCAI" "OOPSLA"	present
"EMNLP" "ICAPS"	**absent**

Table 4.3: **Unlinkable Entities:** Wikipedia contains entries for prominent entities, while missing tail and new entities of the same types.

an entity. In scenarios such as intelligence analysis and local Web search, non-Wikipedia entities are often the most important.

Hence, we extend previous work in entity linking by introducing the *unlinkable noun phrase problem*: Given a noun phrase that does not link into Wikipedia, return whether it is an entity, as well its fine-grained semantic types. Deciding if a non-Wikipedia noun phrase is an entity is challenging because many of them are not entities (*e.g.*, "Some people," "an addition" and "nearly half"). Predicting semantic types is a challenge because of the diversity of entity types in general text. We use the Freebase type system, which contains over 1,000 semantic types. Detection and typing of unlinkable entities can increase yield for NLP applications such as typed question answering.

Our goal is: given (1) a large set of linked assertions L and (2) a large set of unlinked assertions U, for each unlinkable noun phrase subject $n \in U$, predict: (1) whether n is an entity, and if so, then (2) the set of Freebase semantic types for n. For L we use the 9.7 million assertions whose *subject* argument we were able to link in Section 4.1.2, and for U we use the 5 million assertions that we could not link.

We divide the system into two components. The first component (described in Section 4.3) takes any unlinkable noun phrase and outputs whether it is an entity. All $n \in U$ classified as entities are placed in a set E. The second component (described in Section 4.4) uses L and U to predict the semantic types for each entity $e \in E$.

4.2.1 Related Work

Three topics related to the unlinkable noun phrase problem include: NIL features, Named Entity Recognition and Entity Set Expansion.

Previous studies have examined using NIL features to determine whether entity linkers are being asked to link a noun phrase that is not present in Wikipedia [28, 89]. However, there has been no research on whether given noun phrases that are *unlinkable* (for not being in Wikipedia) are entities, and how to make them usable if they are.

Named Entity Recognition (NER) systems can identify named entities within text. For example, given a sentence "Mary Sue woke up," an NER system might identify that "Mary Sue" is an entity and is a *person*. However, a key difference between our final goals and NER is that in the context of entity linking and Wikipedia, we are interested in many more entities than just the *named* entities. For example, "apple juice" and "television" are Wikipedia entities (with Wikipedia articles), but are not traditional named entities. Still, as named entities do comprise a sizable portion of our *unlinkable* noun phrases, we compare against an NER baseline in our entity detection step.

While many NER systems classify entities into 3 types (*person*, *organization* or *location*), there also exist *fine-grained* NER systems [63, 72, 110] that have examined the accuracy loss when scaling NER to more types. On our task, fine-grained NER would suffer similar recall problems as NER because it still does not cover entities that are not named entities. Also, despite including more types, fine-grained NER systems still use an order of magnitude fewer types than we used to cover all entities.

Finally, there is a line of research in using entity set expansion [86] and Web extraction [32] to extract lists of typed entities from the Web (*e.g.*, a list of every city). Our problem instead focuses on determining whether any individual noun phrase is an entity, and what semantic types it holds. Given a noun phrase representing a person name, we return that this is a person entity even if it is not in a list of people names harvested from the Web.

4.3 Detecting the Unlinkable Entities

4.3.1 Introduction

The detection task takes in any *unlinkable* noun phrase and outputs whether it is an entity. There is a long history of discussion in analytic philosophy literature on the question of what exists (*e.g.*, [94]). We adopt a more pragmatic view, defining an "entity" for this task as a noun phrase that could have a Wikipedia-style article if there were no notability or newness considerations, and which would have semantic types. We are interested in entities that could help populate an entity store. "EMNLP 2012" is an example of an entity, while "The method" and "12 cats" are examples of noun phrases that are not entities. This is challenging because at a surface level, many entities and non-entities look similar: "Sex and the City" is an entity, while "John and David" is not. "Eminem" is an entity, while "Youd" (a typo from "You'd") is not.

We address this task by training a classifier with features primarily derived from a timestamped corpus. An intuition here is that when considering only unlinkable noun phrases, usage patterns across time often differ for entities and non-entities. Noun phrase entities that are observed in text going back hundreds of years (*e.g.*, "Europe") almost all have their own Wikipedia entries, so in unlinkable noun phrase space, the remaining noun phrases that are observed in text going back hundreds of years tend to be all the textual references and expressions that are not entities. We plan to use this signal to help separate the entities from the non-entities.

4.3.2 Classifier Features

We use the Google Books ngram corpus [76], which contains timestamped usage of 1-grams through 5-grams in millions of digitized books for each year from 1500 to 2007.[4] We use ngram match count values from case-insensitive matching. To avoid sparsity anomalies we observed in years before 1740, we use the data from 1740 onward. While it has not been used for our task before, this corpus is a rich resource that enables reasoning about knowledge

[4]available at http://books.google.com/ngrams/datasets

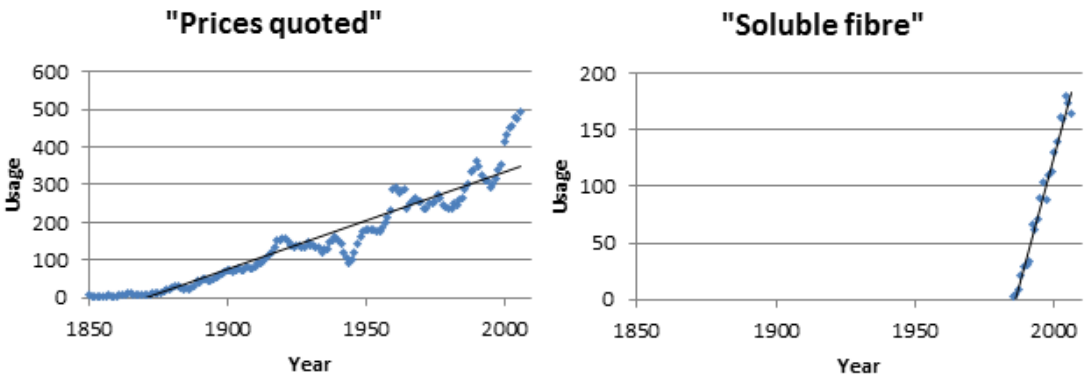

Figure 4.3: **Entity Usage over Time:** Usage over time for the *non-entity* phrase "Prices quoted" (left) increases at lower slope than usage over time for the *entity* phrase "Soluble fibre" (right).

[35] and understanding semantic changes of words over time [125]. Talukdar *et al.* [121] recently used it to effectively temporally scope relational facts.

Our first feature is the *slope* of the least-squares best fit line for usage over time. For example, if a term appears 25 times in books in 1950, 30 times in 1951, ..., 100 times in 2007, then we compute the straight line that best fits {(1950, 25), (1951, 30), ..., (2007, 100)}, and examine the slope. We have observed cases of non-entity noun phrases having lower slopes than entity noun phrases (*e.g.*, Figure 4.3). Note that we do not normalize match counts by yearly total frequency, but we do normalize counts for each term to range from 0 to 1 (setting the max count for each term to 1) to avoid bias from entity prominence. To capture the current usage, in cases where there exists a ≥ 5 year gap in usage of a term we only use the data after the gap.

Another feature is the R^2 fit of the best fit line. Higher R^2 indicates that the data is closer to a straight line. Figure 4.4 plots R^2 vs *Slope* values for some sample noun phrases. We observed that along with their lower *Slope*, the non-entities often also had higher R^2, indicating that their usage does not vary wildly from year to year. This contrasts with certain entities (*e.g.*, "FY 99" for "Fiscal Year 1999") whose usage sometimes varied sharply from year to year based on their prominence in those specific years.

A third feature is *UsageSinceYear*, which finds the year from when a term last started

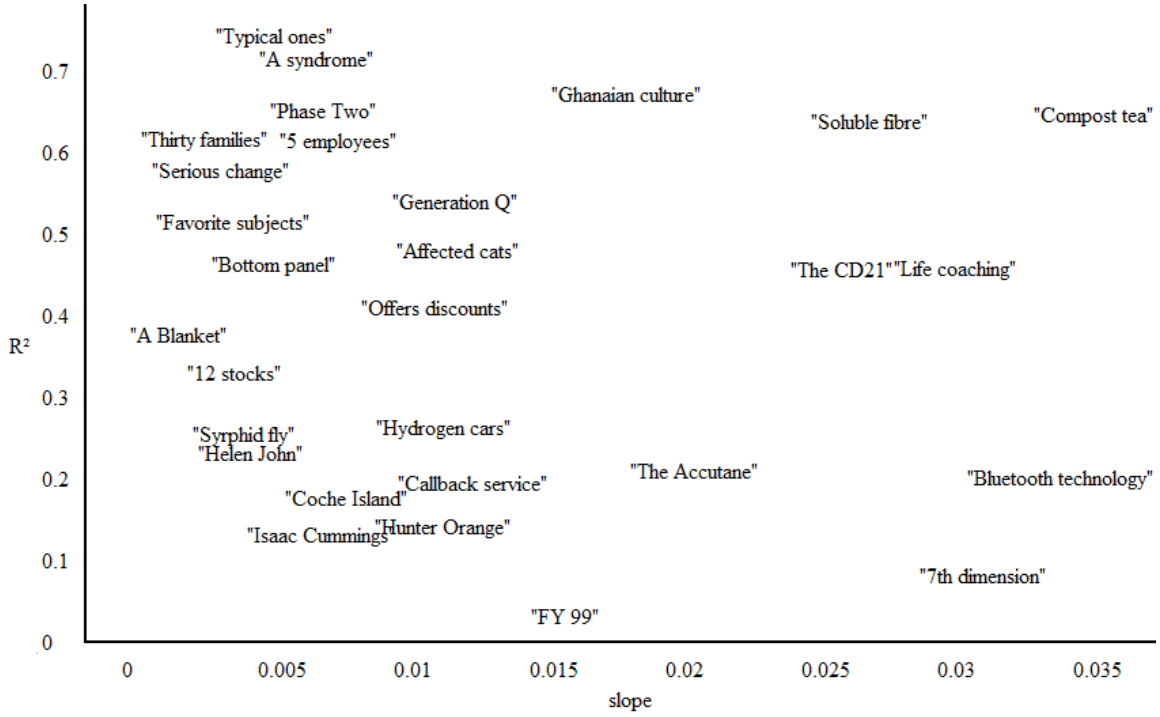

Figure 4.4: **Detecting Entities with Best-Fit Line Data:** Plot of R^2 vs *Slope* for the usage over time of a collection of noun phrases selected for illustrative purposes. Many of the non-entities occur at lower *Slope* and higher R^2, while the entities often have higher slope and/or lower R^2. "Bluetooth technology" actually has even higher slope, but was adjusted left to fit in this figure.

continually being used. For example, a *UsageSinceYear* value of 1920 would indicate that the term was used in books every year from 1920 through 2007. Figure 4.5 shows where various examples terms fall along this dimension.

From the books ngram corpus, we also calculate features for: *PercentProperCaps* - the percentage of case-insensitive matches for the term where all words began with a capital letter, *PercentExactMatch* - the percentage of case-insensitive matches for the term that matched the capitalization in the assertion exactly, and *Frequency* - the total number of case-insensitive occurrences of the term in the book ngrams data, summed across all years, which reflects prominence. Last, we also include a simple *numeric* feature to detect the presence of leading numeric words (*e.g.*, "5" in "5 days" or "Three" in "Three choices").

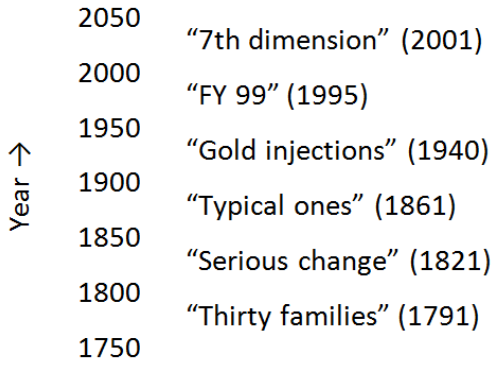

Figure 4.5: **UsageSinceYear of Example Unlinked Terms:** Having appeared in text for a long time but still not having a Wikipedia entry is one potential sign of a non-entity noun phrase.

4.3.3 Evaluation

From the corpus of 15 million ReVerb assertions, there were 1.4 million unlinked noun phrases including 17% unigrams, 51% bigrams, 21% trigrams, and 11% 4-grams or longer.[5] Bigrams comprise over half the noun phrases and the books bigram data is a self-contained download that is easier to obtain and store than the full books ngram corpus, so we focus on bigrams in our evaluation. In a random sample of unlinked bigrams, we found that 73% were present in the books ngram data (65% exact match, 8% case-insensitive match only), while 27% were not (these were mostly entities or errors with non-alphabetic characters). Coverage is a greater issue with longer ngrams (*e.g.*, there are many more possible 5-grams than bigrams, so any individual 5-gram is less likely to reach the minimum threshold to be included in the books data), but as mentioned earlier, only 11% of unlinkable noun phrases were 4-grams or longer.

We randomly sampled 250 unlinked bigrams that had books ngram data, and asked 2 annotators to label each as “entity,” “non-entity,” or “unclear.” Our goal is to separate noun phrases that are clearly entities (*e.g.*, “prune juice”) from those that are clearly not entities (*e.g.*, “prices quoted”), rather than to debate phrases that may be in some entity store

[5]Note that ReVerb already filters out relative pronouns, WHO-adverbs, and existential “there” noun phrases that do not make meaningful arguments.

system	correctly classified
Majority class baseline	50.4%
Named Entity Recognition	63.3%
Slope feature only	61.1%
PUF feature combination	69.1%
ALL features	78.4%

Table 4.4: **Entity Detection Performance:** Our classifier using all features (*ALL*) outperforms majority class and NER baselines.

definitions but not others, so we asked the annotators to choose "unclear" when there was any doubt. There were 151 bigrams that both annotators believed to be very clear labels, including 69 that both annotators labeled as entities, 70 that both annotators labeled as non-entities, and 12 with label disagreement. Cohen's kappa was 0.84, indicating excellent agreement. Our experiment is now to separate the 69 clear entities from the 70 clear non-entities.

For experiment baselines we use the majority class baseline *MAJ*, as well as a Named Entity Recognition baseline *NER*. For *NER* we used the Illinois Named Entity Tagger [97] on the highest setting (that achieved 90.5 F_1 score on the CoNLL03 shared task). *NER* expects a sentence, so we take the longest assertion having the noun phrase. We evaluate several combinations of our features to test different aspects of our system: *Slope* uses only *Slope*, *PUF* uses *PercentProperCaps* + *UsageSinceYear* + *Frequency*, and *ALL* uses all features. We evaluate using the WEKA J48 Decision Tree on default settings, with leave-one-out cross validation.

Table 4.4 shows the results. *MAJ* correctly classifies 50.4% of instances, *NER* correctly classifies 63.3% and *ALL* correctly classifies 78.4%.

4.3.4 Analysis

Overall, 78.4% correctly classified instances is fairly strong performance on this task. By using the described features, our classifier was able to detect and filter many of the non-entity noun phrases in this scenario. Compared to the 63.3% of *NER*, it is an absolute gain of 15.1%, a relative gain of 24%, and a reduction in error of 41.1% (from 36.7% to 21.6%). Student's t-test at 95% confidence verified that the difference was significant.

We found that while low *Slope* (especially with higher R^2) often indicated non-entity, there were numerous cases where higher *Slope* did not necessarily indicate entity. For example, the noun phrase "several websites" has fairly sharp slope, but still does not denote a clear entity. In these cases, the addition of other features can serve as additional useful signal. One error from *ALL* is the noun phrase "Analyst estimates," which the annotators labeled as a non-entity, but which occasionally appears in text (especially titles) as "Analyst Estimates," and is a relatively recent phrase. *NER* misses entities such as "synthetic cubism" and "hunter orange" that occur in our data but are not traditional named entities. We observed that while none of our features achieves over 70% accuracy by themselves, they perform well in conjunction with each other.

4.4 Typing the Unlinkable Entities

If the detection step output that "Sun Microsystems" is likely to be an entity, then the next step is to further predict that it has the Freebase types such as *organization* and *software developer*. Our typing will use a set of linked assertions L and set of unlinked assertions U to predict the semantic types for each unlinked entity $e \in E$ from U.

Each linked entity in L occurs with a set of relation phrases. For example, L might contain that the entity *Microsoft* links to a particular Wikipedia article, and also that it occurs with relation phrases such as "has already announced" and "has released updates for." For each Wikipedia-linked entity in L, we further look up its exact set of Freebase types. Each unlinked entity $e \in E$ also occurs with a set of relation phrases (from U). We now have a large set of class-labeled instances (all entities in L), a large set of unlabeled instances (E), and a method to connect the unlabeled instances with the class-labeled instances (via any

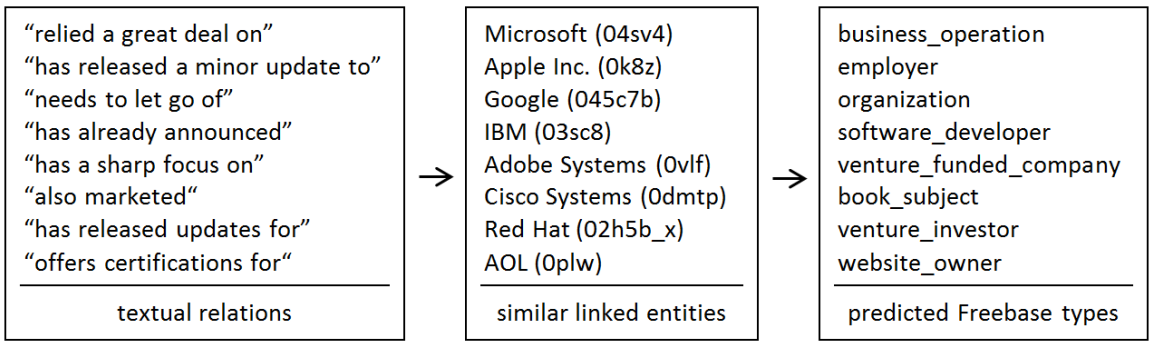

Figure 4.6: **Predicting Entity Types:** This example illustrates the set of Freebase type predictions for the noun phrase "Sun Microsystems." We predict the semantic type of a noun phrase by: (1) identifying the relation phrases that occur with this noun phrase as the first argument, (2) identifying the linked entities that also appear in the domains of those relation phrases, and (3) observing their semantic types.

shared relation phrases), so we cast this task as an instance-to-instance class propagation problem [59] for propagating class labels from labeled to unlabeled instances.

We build on the recent work of Kozareva *et al.* [59], and adapt their approach to leverage the scale and resources of our scenario. While they use only one type of edge between instances, namely shared presence in the high precision *DAP* pattern [48], our final system uses 1.3 million relation phrases from our extractions, corresponding to 1.3 million potential edge types. Their evaluation involved only 20 semantic classes, while we use all 1,339 Freebase types covered by our entities in L.

There is a rich history of other approaches for predicting semantic types. One approach is to model relationships between instances and classes [119, 120], but the unlinked entities do not come with any class information. Pattern-based approaches [84, 88] are popular, but [59] notes that they are "constraint to the information matched by the pattern and often suffer from recall," meaning that there are many instances they miss. Classifiers have been trained for fine-grained semantic typing, but only with far fewer types. Rahman and Ng [96] studied hierarchical and collective classification using 92 types, and Ling and Weld's FIGER system [72] recently used an adapted perceptron for multi-class multi-label classification into 112 types.

4.4.1 Algorithm

Given an entity e, our algorithm involves: (1) finding the relation phrases that have e in their domain, (2) finding linked entities that are also in the domain of those relation phrases, and then (3) predicting types by observing the types of those linked entities. The same technique can also be run with relation range instead of relation domain. Figure 4.6 illustrates how we would predict the semantic types of the noun phrase "Sun Microsystems."

Find Relations: Obtain the set R of all relation phrases in U that have e in their domain. For example, if U contains the assertion "(Sun Microsystems, has released a minor update to, Java 1.4.2)," then the relation phrase "has released a minor update to" should be added to R when typing "Sun Microsystems."

Find Similar Entities: Find the linked entities in L that are in the domain of the most relations in R. In our example, entities such as "Microsoft" and "Apple Inc." have the greatest overlap in relation phrases because they are most often in the domain of the same relation phrases, *e.g.*, ("Microsoft, has released a minor update to, Windows Live Essentials"). Create a set S of the entities that share the most relation phrases. We found keeping 10 similar entities ($|S| = 10$) is generally enough to predict the original entity's types in the final step.

Predict Types: Return the most frequent Freebase types of the entities in S as the prediction. To avoid penalizing types containing very few entities (*e.g.*, if a type has only 7 entities total then it cannot make up all 10 similar entity spots), if there are n instances of semantic class C in S, then we rank C using a type score $T(n, C, S) = max(n/|S|, n/|C|)$. We found this to perform better than $T(n, C, S) = avg(n/|S|, n/|C|)$. For "Sun Microsystems," *business operation* was the top predicted type because all entities in S were observed to include *business operation* type.

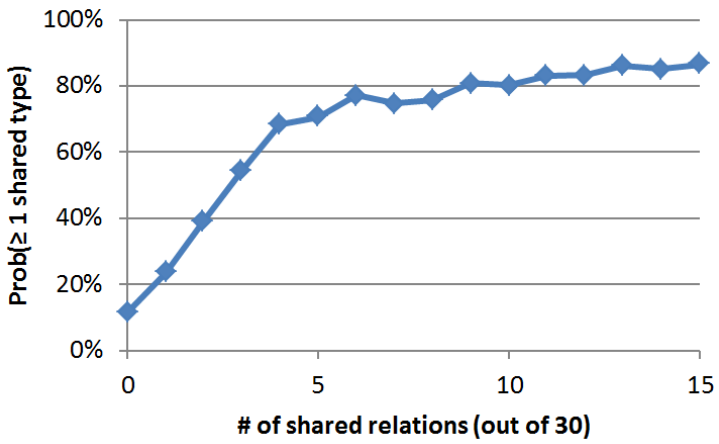

Figure 4.7: **Effect of Shared Relation Phrases:** Entities that share more relation phrases are more likely to have semantic types in common.

4.4.2 Edge Validity

This algorithm will only be effective if entities that share relation phrases are more likely to be of the same semantic types. To verify this, we sampled 30,000 linked entities from L that had at least 30 relation phrases each, and associated each with their 30 most frequent relations. From the 900 million possible entity pairs, we then randomly sample 500 entity pairs that shared exactly k out of 30 relations, for each k from 0 to 15. At each k we then use our sampled pairs to estimate the probability that any two entities sharing exactly k relations (out of their 30 possible) will share at least one type.

The results are shown in Figure 4.7. We found that entities sharing more relation phrases were in fact more likely to have semantic types in common. Two entities that shared exactly 0 of 30 relation phrases were only 11% likely to share a semantic type, while two entities that shared exactly 10 of 30 relations were 80% likely to share a semantic type. This validates our use of relation phrases as a signal-bearing edge in instance-to-instance class propagation.

4.4.3 Weighting Relation Phrases

The algorithm as currently described treats all relation phrases equally, when in practice some are stronger signal to entity type than others. For example, two entities in the domain

of the "came with" relation often will not share semantic types, but two entities in the domain of the "autographed" relation will almost always share a type.

One way to capture this intuition could be to use frequency ratios (analogous to TFIDF weighting) over relations phrases to determine which relation phrases are more meaningful when observed. Across a large entity-linked corpus, we can count the total number of different entities or types that appear with each relation phrase. A relation phrase such as "autographed" will only have *people* types in its domain, while a relation phrase such as "came with" will appear with *people*, *vehicles*, *toys*, *food items*, and many other types. A metric such as *(total # of types in corpus)* divided by *(# of types observed with a relation phrase)* would correspond to how useful a relation phrase is for finding entities of the same type. At 1,339 total types, if "autographed" appeared with 7 types then it would have a weight of 191.3. If "came with" appeared with 32 types, then it would have a lower weight of 41.8.

While this metric is useful, it does not account for real world distributions of types in the domains of relation phrases. For example, consider a relation R_1 whose domain is 99% *people* and 1% *locations*, and another relation R_2 whose domain is 50% *people* and 50% *locations*. R_1 and R_2 both appear with exactly 2 types and would have the same weight based on our described metric. However, if we are trying to find known entities that share a type with a new entity which occurs with both R_1 and R_2, then sharing R_1 turns out to be a much more informative signal than sharing R_2. This is because two random entities from the domain of R_1 are more likely to share a type than two random entities from the domain of R_2.

The more precise way to account for how useful a relation will be for finding entities that share a type is to directly define relation weight $w(r)$ as the observed probability (among the linked entities) that two entities will share a Freebase type if they both occur in the domain of r. This version of relation weight would calculate from our observed data that two random entities in the domain of R_1 will share a type 98.02% of the time and assign it a weight of 0.9802, and that two random entities in the domain of R_2 will share a type 50% of the time and assign it a weight of 0.5. Formally, if $D(r)$ = all entities observed in the domain of relation r and $T(e)$ = all Freebase types listed for entity e, then weight $w(r)$

"is a highway in"
"is a university located in"
"became the president of"
"turned down the role of"
"has an embassy in"

Table 4.5: **High-Weight Relations:** High-weight relations provide a strong signal when finding entities likely to share a type.

"comes with"
"is a generic term for"
"works best on"
"can be made from"
"is almost identical to"

Table 4.6: **Low-Weight Relations:** Low-weight relations do not help much in finding entities that share types.

of a relation phrase r is:

$$w(r) = \sum_{e_1, e_2 \in D(r),\ e_1 \neq e_2} \frac{I(e_1, e_2)}{|D(r)| \cdot (|D(r)| - 1)} \tag{4.5}$$

$$I(e_1, e_2) = \begin{cases} 1, & \text{if } |T(e_1) \cap T(e_2)| > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

We use this technique to calculate weights for all relation phrases in L. Table 4.5 shows examples of high weight relations, and Table 4.6 shows low weight relations. We now modify the *Find Similar Entities* step such that if a linked entity shares a set of relations Q with the entity being typed, then it receives a score which considers all shared relations $q \in Q$ but uses the high weight relations more. On a development set we found that a score of $\sum_{q \in Q} 10^{4 \cdot w(q)}$ was effective, as higher weight signifies much stronger signal. This score then determines which entities to place in S.

	HEAD		TAIL	
	Prec@1	**F_1**	**Prec@1**	**F_1**
$\mathbf{B_{Random}}$	0.008	0.028	0.004	0.023
$\mathbf{B_{Frequency}}$	0.244	0.302	0.298	0.322
$\mathbf{S_{NoWeight}}$	$0.542^{\dagger}$	$0.465^{\dagger}$	$0.510^{\dagger}$	$0.456^{\dagger}$
$\mathbf{S_{Weighted}}$	$0.610^{\ddagger}$	$0.521^{\ddagger}$	$0.598^{\ddagger}$	$0.522^{\ddagger}$

Table 4.7: **Evaluation of Type Predictions:** The top type predicted by our $S_{Weighted}$ method is correct about 60% of the time, while the top type predicted by the $B_{Frequency}$ baseline is correct under 30% of the time. † indicates statistical significance over $B_{Frequency}$, and ‡ over both $B_{Frequency}$ and $S_{NoWeight}$.

4.4.4 Evaluation

The goal of the evaluation is to judge how well our method can predict the Freebase semantic types of entities in our scenario. Our linked entities covered 1,339 Freebase types, including many diverse types ranging from *computer operating systems* to *airlines*, from *baseball teams* to *religious texts*. Human judges would have difficulty internalizing the specific characteristics of so many types and keeping them all in mind while manually annotating new entities. Instead, we automatically generate testing data by sampling entities from L, and then test on ability to recover the actual Freebase types (which we know).

We sample a *HEAD* set of distinct 500 Freebase entities (drawn randomly from our set of linked extractions), and a *TAIL* set of 500 entities (drawn randomly from our set of linked entities). An entity that occurs in many extractions is more likely to be in *HEAD* than *TAIL*. Our sampling also picks only entities that occur with at least 10 relations, which is appropriate for the Web scenario where we can just query to obtain more instances.

For baselines we use a random baseline B_{Random} and a frequency baseline $B_{Frequency}$ which always returns types in order of their frequency among all linked entities (*e.g.*, always *person*, then *location*, *etc*). We evaluate our system without relation weighting ($S_{NoWeight}$) and also with relation weighting ($S_{Weighted}$). For $S_{Weighted}$ we leave all the test set entities out when calculating global relation weights. Our metrics are *Precision at 1* and F_1 *score*. Precision at 1 measures how often the top returned type is a correct type, and is useful for

applications that want one type per entity. F_1 measures how well the method recovers the full set of Freebase types (for each test case we graph precision/recall and take the max F_1), and is useful for applications such as typed question answering.

Table 4.7 shows the results. B_{Random} performs poorly because there are so many semantic types, and very few of them are correct for each test case. $B_{Frequency}$ performs slightly better on *TAIL* than *HEAD* because *TAIL* contains more entities of the most frequent types. $S_{NoWeight}$ performance is statistically significant above all baselines, and $S_{Weighted}$ is statistically significant over $S_{NoWeight}$ on both test sets and metrics. Significance is measured using the Student's t-test at 95% confidence.

4.4.5 Analysis

$S_{Weighted}$ was successful at recovering the correct Freebase types of many entities. For example, it finds that "Atherosclerosis" is a *medical risk factor* by connecting it to "obesity" and "diabetes," that "Supernatural" is a *TV program* and a *Netflix title* by connecting it to "House" and "30 Rock," and that "America West" is an *aircraft owner* and an *airline* by connecting it to "Etihad Airways" and "China Eastern Airlines."

One example where $S_{Weighted}$ predicted top types that did not match with Freebase is *fictional characters*. Many *fictional characters* participate in a relation phrases that make them look like *people* (*e.g.*, "was born on"), but predicting that they belong to *people* class is incorrect because Freebase does not consider them to be *people*. Some performance hit was also due to entity linking errors. From an assertion like "The Four Seasons is located in Hamamatsu," our entity linker (and other entity linkers we tried) prefer linking "The Four Seasons" to Vivaldi's music composition rather than the hotel chain. This can then lead our type prediction algorithm to sometimes choose "The Four Seasons" (*music composition*) as a similar entity to new entities that occur with "is located in."

As a general reference for performance of state-of-the-art fine-grained entity classification, the FIGER system [72] for classifying into 112 types reported F_1 scores ranging from 0.471 to 0.693 in their experiments. It is important to note that these numbers are not directly comparable to us because they used different data, different (and fewer) types, and

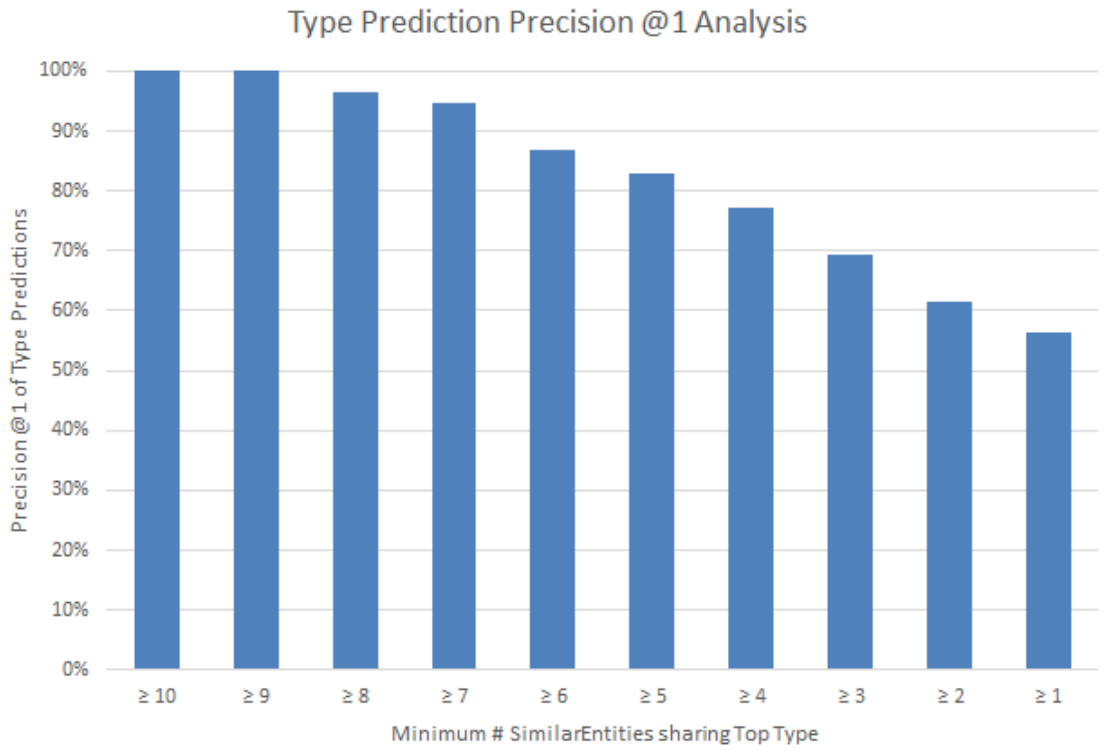

Figure 4.8: **Type Prediction Tiers:** Predictions where most of the *similarEntities* share a type are more likely to be correct. While overall precision at 1 is around 60%, the predictions where ≥ 7 *similarEntities* share a type have over 90% precision.

different evaluation methodology.

While precision at 1 around 60% may not be high enough yet for certain applications, it is significantly better than our baselines approaches, which are under 30%, and we hope that our values can serve as a useful reference point on this task for future systems. For applications that need higher typing precision and do not need 100% recall, we can order the type predictions by how likely they are to be correct. One way to do this is to sort the type predictions by the *number of similar entities with that type*. If we are predicting types for "diet Cherry coke" and 9 out of 10 similar entities are *beverages* while 4 out of 10 similar entities are *business brands*, we are more confident that "diet Cherry coke" is a *beverage* than a *business brand*. This intuition could also be applied across entities. If 6 out of 10 similar entities for "America West" are *aircraft owners*, then we are more confident

that "diet Cherry coke" is a *beverage* than we are that "America West" is an *aircraft owner*, but less confident that "diet Cherry coke" is a *business brand* than we are that "America West" is an *aircraft owner*.

If we take only the type predictions where most of the similar entities share some type, then precision is significantly higher. We obtained 150,000 unlinkable noun phrases which each had at least 5 relations and did not begin with words often associated with references or multiple entities (*e.g.*, "a", "the", "two").[6] We randomly sampled 200 of these, and manually labeled the top type predictions for each: there were 98 good predictions, 76 bad, and 26 which we discarded for being non-entity or for only having trivially true type predictions such as *book subject*. While Precision @1 of the remaining 174 entities is only around 60%, it becomes much higher when restricting to just the predictions where many similar entities shared the top type. Figure 4.8 shows that when at least 4 similar entities all shared the top type, 75/97 (77%) of the top predictions were correct. With 6 similar entities this increased to 47/54 (87%), and at 8 similar entities this increased to 28/29 (97%). For applications that benefit from any accurate typing of unlinkable entities and do not require 100% recall, setting these thresholds can provide high-precision subsets of the data.

Splitting the data by number of entities sharing a top type reveals not only when the type predictions are more likely to be correct, but also reveals when the type predictions are more likely to be incorrect. 3/4 of incorrect type predictions occurred when ≤ 3 similar entities shared a top type. Incorrect type predictions here were primarily due to entities not occurring in the data with enough useful relations to get a clear signal on the type. For example, if an unlinked entity occurs only with relation phrases such as "could be", "are no longer in", or "seem to be like", then no clear types will emerge from the similar entities. 1/4 of incorrect type predictions occurred when ≥ 4 similar entities shared a top type. Of these cases, 33% were due to the presence of misleading relations. For example, if we see "RailsConf will be heading to Las Vegas" then we might incorrectly predict that "RailsConf" is a *person* because "will be heading to" usually occurs with people (*e.g.*, Serena Williams). 27% were due to entity types beyond our types inventory (*e.g.*, "$400m" does

[6]Full entity detection is not used in this evaluation because that had only been set up with bigrams, and adding in the larger ngram sets required more resources than were available at the time.

not match any of our types), 20% were due to extraction errors, 13% were due to relation sparsity, and 7% were due to error propagated from the linking component.

Because the majority of error involved cases where an entity did not occur with enough useful relation phrases to make a good type prediction, one future work direction here is to incorporate type propagation signals beyond shared relation phrases. For example, shared term heads can also help with certain entity types. If we observe a new entity "Endolyne Park" and it only occurs with a few low-weight relations, we can still type it accurately if we observe that it ends on the word "Park" and select a set of similar entities that also end on the word "Park." Beyond this, our error analysis also indicates that performance might improve if we refine our types set and further improve extraction and linking accuracy.

4.4.6 Discussion

Together, sections 4.3 and 4.4 presented an approach for working with non-Wikipedia entities in text. Consider the following possibilities for a noun phrase in a text corpus:

Wikipedia Entity: (*e.g.*, "Computer Science," "South America," "apple juice") - Entity linking techniques can identify and type these.

Non-Wikipedia, Non-Entity: (*e.g.*, "strange things," "Early studies," "A link") - Our classifier from Section 4.3 is able to filter these.

Non-Wikipedia, Entity: (*e.g.*, "Safflower oil," "prune juice," "Amazon UK") - We identify these as entities, then propagate semantic types to them. Our technique finds that "Safflower oil" occurs with high weight relations such as "is sometimes used to treat" and "can be substituted for," making it similar to linked entities such as "Phentermine" and "Dandelion," and then correctly predicts semantic types including *food ingredient* and *medical treatment*. Appendix D contains a number of additional examples of non-Wikipedia entities in our Web extraction data that we automatically typed.

4.4.7 Conclusions

We showed that while entity linking cannot link to entities outside of Wikipedia, once a large text corpus has been entity linked, the presence and content of the existing links can be leveraged to help detect and semantically type the non-Wikipedia entities as well. We designed techniques for detecting whether unlinkable noun phrases are entities. We further propagate semantic types from our linked entities to any unlinkable noun phrase entities. In our evaluations, we showed that our techniques achieve statistically significant improvement over appropriate baselines.

4.5 The Challenge of Splitting Ambiguous Surface Strings

4.5.1 Motivation

The previous section implicitly assumed that if a noun phrase did not link into our knowledge base, then it represented at most one entity. For example, if the string "Mausam" was not in our knowledge base, then we might identify that (1) it was an entity, and then (2) predict that it referred to a *person*. But in reality there exists a separate "Mausam" *person*, "Mausam" *film* and "Mausam" *journal*. Instead of outputting that "Mausam" is a single noun phrase which holds all of those types concurrently, a stronger output would be to realize the ambiguity, determine how many senses there are, and determine which sense is being referred to in each instance. In a random sample of 100 unlinked entities (that each occurred with at least 5 relations), when going through them manually we found that 34 of them had names shared by different entities on the Web.

In this section we present how using the idea of *mutual exclusion* [17] can help us detect this type of *surface string ambiguity*, assuming that the type prediction step was accurate. For example, if we predict that a noun phrase has *person* and *film* types, but we also observe in our linked instances that these types never coexist within a single entity, then this is good evidence that the noun phrase refers to multiple entities.

4.5.2 Mutual Exclusion Constraints

The set of 15 million extractions that we linked into Wikipedia covered 353,170 distinct entities and 1,339 Freebase types. Looking up the types of each of the entities, we can observe that there are 125,660 *people* (35.6% of the entities), 78,674 *locations* (22.3%), 39,445 *organizations* (11.2%), 36,319 *employers* (10.3%), *etc.* We can then calculate the number of entities that we would expect to share each pair of types if all types were independent. For example, if the *person* and *location* types were independent, then we would expect 353,170 * 35.6% * 22.3% = 27,992 of our linked entities to be both *person* type and *location* type. However, it turns out that only 30 of our linked entities are labeled in Freebase as being both a *person* and a *location*. This is significantly fewer than we expect, so it tells us that *person* and *location* types are not independent. Instead, they are likely *mutually exclusive*, meaning that they tend not to occur with each other within single entities.

This same method can also identify *type affinity*, meaning types that tend to occur together more often than random. For example, if the *organization* and *employer* types were independent then we would expect 353,170 * 11.2% * 10.3% = 4,056 of our entities to be both *organization* and *employer* type. However, 30,778 of our entities have both *organization* and *employer* types, which is significantly more than expected. We can then say that *organization* and *employer* types have type affinity.

Given a type pair (t_1, t_2), we can now identify *mutual exclusion* and *type affinity* relationships between t_1 and t_2 if the types are prominent enough. Note that this does not work if both t_1 and t_2 are sparse types. For example, if only 0.2% of our entities have type t_1 and 0.1% of our entities have type t_2, then we would expect 353,170 * 0.2% * 0.1% = 0.7 entities in our set to be of both type t_1 and t_2. This number is so small that even if we observe no entities sharing (t_1, t_2), this is within the margin of error and is not strong enough evidence to say that t_1 and t_2 are mutually exclusive. Thus, we set a minimum threshold on the number of elements that we would expect to have both types, if they were independent.

Setting a threshold of 10 elements, we find that there are 11,052 pairs of types where we would expect at least 10 elements to be of both types, if the types were independent.

Of these pairs, 6,284 (57%) have <1% as many actual elements of both types, compared to if they were independent. 5,984 (54%) have no actual elements of both types. This means that just over half of our type pairs are quite likely to be mutually exclusive. For example, we find that *person* type is mutually exclusive with types such as *architectural structure*, *building*, *body of water*, *disease*, *river* and *airport*. We can also easily read off the types that have the most affinity with any given type. *person* type has the most affinity with types such as *baseball player*, *congressperson*, *basketball player*, *theater actor*, *cricket player* and *athlete*.

4.5.3 Graph Coloring

Once we have the set of mutual exclusion constraints, the problem of splitting apart the senses can be formulated as a *graph coloring* [52] problem. Given a set of predicted types and a set of mutual exclusion constraints between these types, we set up a graph G where each type is a vertex and each mutual exclusion constraint is an edge. For example, if we predict that "Mausam" is a *person*, *film*, *journal* and a *professor* and we know that (1) no *people* are *films*, (2) no *people* are *journals*, and (3) no *films* are *journals*, then we would set up a graph with 4 vertices (1 per type) and 3 edges (1 per constraint).

Now, the problem of splitting apart the ambiguous senses of the string becomes the problem of assigning a *color* to each vertex of the graph such that no adjacent vertices share the same color. For example, if we color the *person* node red, then the *film* and *journal* nodes must be a different color because they share an edge with the *person* node. One coloring that satisfies these constraints would be to color *person* and *professor* with red, color *film* with blue, and color *journal* with green. This coloring would then correspond to saying that the string "Mausam" has 3 different senses including a "red" sense that is a *person* and a *professor*, a separate blue sense that is a *film*, and a separate green sense that is a *journal*. This method of dividing a string into senses can determine when a string is ambiguous (when it contains *mutually exclusive* types), and then split the string into internally consistent *senses* that do not violate any mutual exclusion constraints.

To avoid having multiple senses that are actually the same, it is desirable to color the

graph using as few colors as possible. If we did not use the minimal number of colors then we might color the *person* and *professor* senses differently, and then predict that there are 4 senses of "Mausam" in our data when there are actually only 3. The minimum number of colors needed to color a graph is referred to as the *chromatic number* [31]. For our problem, the chromatic number corresponds to the minimum number of separate senses that need to be set up to satisfy all the mutual exclusion constraints.

For nodes that can take multiple potential colors without violating any constraints, we use the type affinity values to determine node color. For example, *professor* will have more affinity with *person* than the other types, so we assign it to *person* type. If a node can take color c_1 or c_2 and has affinity to existing vertices in both colors, then we can assign it to both colors. This corresponds to cases such as having a *book subject* type that all distinct senses can share.

While graph coloring is a computationally hard problem and finding chromatic number is NP-complete, it has been studied extensively for use in problems such as register allocation [20], and there are numerous practically efficient implementations available, especially for solving smaller problems. When there are many potential types available, we can bound the problem by limiting the number of vertices that we admit to the graph to only the most probable types.

4.5.4 Evaluation

As an initial evaluation of this technique, we selected 10 common ambiguous surface strings from Freebase that each had 2 distinct senses, including "Titanic" (*film* vs *ship*), "Paris" (*city* vs *person*), "Amazon" (*company* vs *rainforest*), "Seattle" (*city* vs *football team*), "Apple" (*fruit* vs *company*), "Georgia" (*country* vs *state*), "Nirvana" (*band* vs *belief*), "Minesweeper" (*ship* vs *game*), "Firefly" (*tv series* vs *insect*) and "Phoenix" (*city* vs *bird*). These 20 senses covered 169 types because an individual sense can have multiple types (*e.g.*, "Paris" the *city* is also a *location*, *employer*, *olympic host city*, *etc*). For each surface string we then combined all possible types from the 2 senses, and evaluated on whether our technique could accurately split apart the 2 senses. We set a mutual exclusion threshold at

10%, meaning that we deemed a pair of types to be mutually exclusive if we observed fewer than 10% the number of entities we would expect if the types were independent.

On 7 of the 10 strings, our method was able to detect that there were 2 necessary senses, and divide all the possible types into 2 separate clusters using graph coloring. It worked best for cases like "Paris" where both senses have very prominent types (*e.g.*, *city* vs *person*), and was able to identify that one sense had types such as *location*, *employer*, *dated location*, *statistical region*, *etc.*, while the other sense had types such as *person*, *celebrity*, *tv personality*, *social network user*, *etc.* Our method did not work for cases like "Minesweeper" where the senses (*ship* and *game*) were all sparse enough that no mutual exclusion constraint could be detected between them, as described earlier. We also noticed some error due to the mutual exclusion threshold value. Setting it too low would cause the technique to not separate many pairs of types that should not occur together, but setting it too high causes pairs of non-mutually exclusive types to be identified as mutually exclusive.

4.5.5 Discussion

In cases where an unlinkable noun phrase is actually ambiguous and contains multiple mutually exclusive senses, the technique described in this section is able to detect that multiple senses are needed and identify which of the potential types correspond to each of the necessary senses. However, some challenges remain. First, the technique described relies on higher accuracy type predictions than are currently available. This should work out over time as we improve entity typing accuracy. Second, there will also exist cases where a noun phrase is ambiguous even within types (*e.g.*, the string "John Smith" can refer to many different *people*). In those cases we would occasionally be able to detect ambiguity due to violation of functionality constraints (as described in Section 3). For example, if we see ("John Smith", "was born in", "Seattle") and also ("John Smith", "was born in", "Boston"), then this would be strong evidence that the string was ambiguous. There will also be many cases on the Web where there is insufficient knowledge present (even for humans) to detect the ambiguity. Correct handling of ambiguity in unlinkable noun phrases on the Web is an area where much future work remains to be done.

	top assertions
rank by freq	"(teachers, teach at, school)"
	"(friend, teaches at, school)"
	"(Mike, teaches at, school)"
	"(biologist, teaches at, Harvard)"
	"(Jorie Graham, teaches at, Harvard)"
rank by link score	"(Pauline Oliveros, teaches at, RPI)"
	"(Azar Nafisi, teaches at, Johns Hopkins)"
	"(Steven Plaut, teaches at, Univ of Haifa)"
	"(Niall Ferguson, teaches at, NYU)"
	"(Ha Jin, teaches at, Boston University)"

Table 4.8: **Improved Extraction Ranking:** Ranking based on *link score* gives higher quality results than ranking based on *frequency*.

4.6 Applications

While we began with the motivation of exploring what additional information could be learned for extraction arguments, the steps covered in this chapter also make Open IE Web extractions usable for many more tasks than previously possible. Entity linking (Section 4.1) has applications such as improving question answering ranking functions and generating selectional preferences knowledge for relation phrases. Combining linking with typing for unlinkable entities further improves upon the experience for typed question answering tasks and inference tasks.

4.6.1 Improved Instance Ranking Function

We observed our *link score* to be a better ranking function than *assertion frequency* for presenting query results. For example, Table 4.8 shows the top results when searching the extractions for instances of the "teaches at" relation phrase. When results are sorted by frequency in the corpus, assertions like "(friend, teaches at, school)" and "(Mike, teaches at, school)" are returned first. When results are sorted by *link score*, the top hundred results

are all specific instances of professors and the schools they teach at, and are noticeably more specific and generally correct than the top frequency-sorted instances. This also suggests that link score could serve as additional signal for our *specific* filter in Section 2.3.2.

4.6.2 Freebase Selectional Preferences

Over 15 million extractions, the entities that we linked to covered over 1,300 Freebase types. Knowing these entity types then allows us to compute the *Freebase selectional preferences* for each of our 1 million+ relation phrases. For example, we can observe from our linked entities that the "originated in" relation most often has types such as *food*, *sport*, and *animal breed* in the domain. Selectional preferences have been calculated for WordNet [1], but have not been calculated at scale for Freebase, which is something that we get for free in our scenario. This selectional preference information can potentially be used to improve our entity linking process. For example, if we see "Maize" as a first argument to "originated in" then the selectional preferences tell us that it is more likely to be referring to the *food* sense of "Maize" than the *color* sense. It could also have applications in other systems that use Freebase types.

4.6.3 Typed Question Answering

From our set of 15 million assertions, we found and typed many non-Wikipedia entities. In *food* while Wikipedia has "crab meat," we find it is missing others such as "rabbit meat" and "goat milk." In *job titles* it has "scientist" and "lawyer," but we find it is missing "PhD student," "fashion designer," and others. We find many of the *people* and *employers* not prominent enough for Wikipedia.

One application of this research is to increase the yield of applications such as Typed Question Answering [16]. For example, consider the query "What *computer game* is a lot of fun?" A search for assertions matching "* is a lot of fun" in the data yields around 1,000 results such as "camping," "David Sedaris" and "Hawaii." Entity linking allows us to identify just the *computer games* in Wikipedia that match the query, such as "Civilization." However, around 400 query matches could not be entity linked. Our noun-phrase classifier

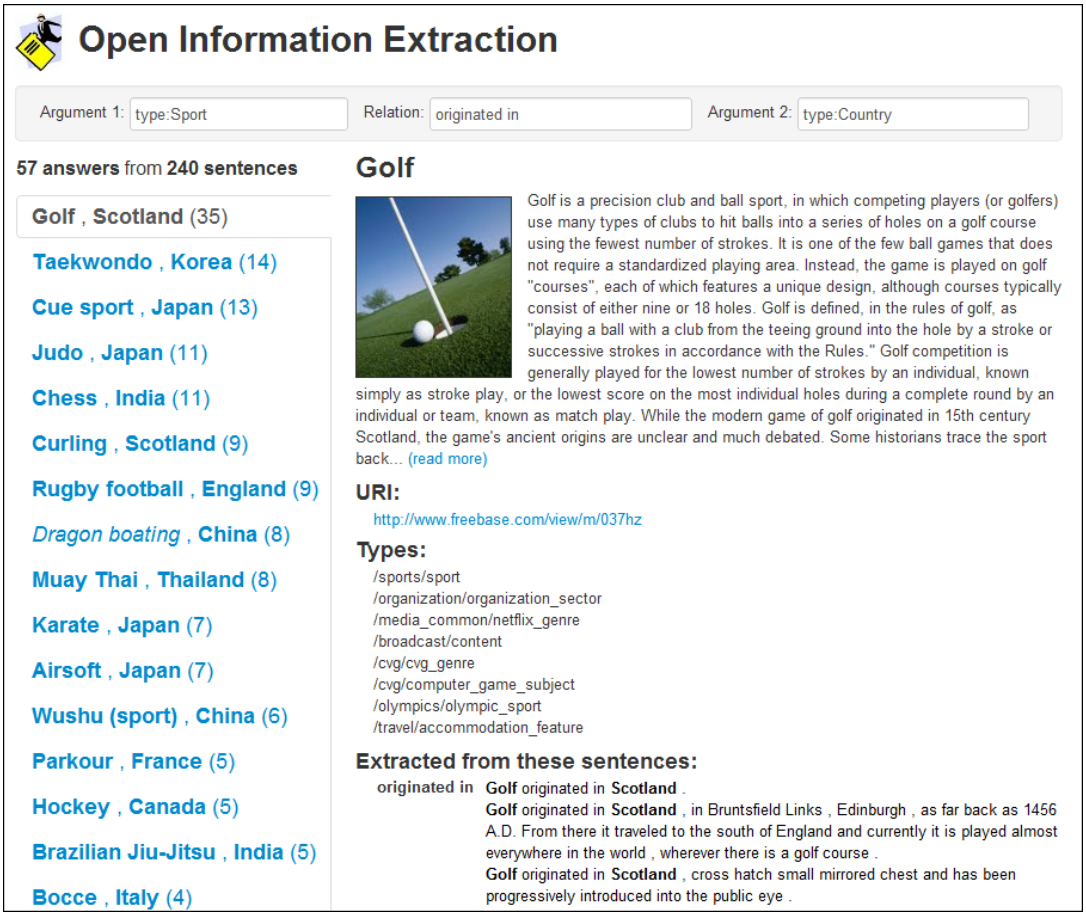

Figure 4.9: **Typed Question Answering:** After linking, we can answer *typed queries* such as "Where did different sports originate?" Linked Freebase information allow us to set up a richer experience with images and blurbs. Unlinkable entities such as "Dragon Boating" have their types predicted.

filters out non-entities such as "actual play," "Just this" and "Two kids." After predicting types for the matches that did not get filtered, we find additional non-Wikipedia *computer games* that match the query, including "Cooking Dash," "Delicious Deluxe" and "Slingo Supreme."

Figure 4.9 further displays the type of interface we have set up now after all the linking steps.[7] We can answer typed queries such as "Which sports originated in which countries?",

[7]Screenshot from the demo at http://openie.cs.washington.edu/

and understand which of our extractions involve *sports* and *countries*. The links into Freebase also enable us to draw in additional data such as images and blurbs to enable a richer user experience than the original interface shown in Figure 2.1. "Dragon Boating" (middle left in Figure 4.9) is not its own entity within Wikipedia and Freebase because it is discussed within the "Dragon Boat" (*ship*) article. However, our unlinkable entity detecting and typing steps allow us to detect that it is a separate entity, and furthermore that it is a *sport* and should be included in queries for *sports*.

4.6.4 Inference

Typed and disambiguated entities are especially valuable for inference applications over extracted data. For example, if we observe enough instances like "Orange Juice is rich in Vitamin C," "Vitamin C helps prevent scurvy," and "Orange Juice helps prevent scurvy," then we can learn the inference rule that if a *beverage* is rich in a *nutrient* and that *nutrient* helps prevent a particular *disease*, then the *beverage* also helps prevent the *disease*. Schoenmackers *et al.* [107] explored this, but without entity linking they had to rely on heavy filtering against hypernym data, losing most of their extraction instances in the process.

Other tasks that involve reasoning over extractions, such as learning relation properties (Chapter 3) and generalizing instances to conceptual knowledge [21], would also all benefit from using linked instances rather than the noisy plain Web extractions.

4.6.5 Notes on Scalability

Bart [7] recently applied the linking and typing techniques discussed in this chapter to a set of 1 billion ReVerb extractions. A distributed implementation of the linking code was able to link 500 million extractions in 3 days time. The 15 million extractions we reported results on are a high quality subset of the 1 billion, so the 1 billion contains a large number of noisier and lower quality extractions as well. Both the linking and typing components include parameters that can be tuned to trade off precision with yield to help address problems with lower extraction quality. For the linking component, Bart found that setting a link score threshold of 5 gave accurate enough results in practice to include in a user-facing

interface. For the typing component, requiring at least 6 similarEntities share any predicted type and only using relations with weights of at least 0.33 gave good precision, but these settings could be modified to increase yield. Current yield for the typing was 1 million arg1 strings and 1 million arg2 strings typed (out of 40 million / 90 million respectively).

The high-quality typed question answering capability shown in Figure 4.9 is currently powered by data from this effort. To evaluate the quality of linking and typing in this demo, we randomly sampled 20 of the queries that Web users issued to this demo between June and October 2012 which had at least 100 answers each. We then manually reviewed the top 10 answers the demo returned for each query. Of the 41 argument strings that had entity links, 95% were linked correctly. Of the 37 argument strings that had type predictions, 74% had a correct top predicted type. For the most part, errors were understandable (*e.g.*, linking "respiratory depression" to "clinical depression" or predicting that "allergic reactions" is a disease). Some of the links were fairly good, *e.g.*, knowing that "Super Bowl" in an assertion about the Seahawks had to mean "Super Bowl XL," and that "radiation" when used with "Gamma Rays" means "Electromagnetic radiation." Some extraction arguments that are not unique entities could still be correctly typed. For example, the typing recognizes that the string "his father" is referring to a *person*.

Two further areas of future work suggested by the work on 1 billion extractions are the use of Web-scale alias lists in entity linking and further refinement of the types used. This larger data set includes many more uncommon aliases of known entities than the set of 15 million extractions, so a Web-scale alias list such as the 300 million string-to-concept mappings from [116] could help increase yield and allow more arguments to be linked. Also, while the Freebase type system has many advantages over other type systems, further work remains on how the exact types inventory could be best adjusted given specific scenarios. In user-facing settings such as question answering, we found that Freebase contains some types (*e.g.*, *dated location*) that are rarely useful to users, while missing some other types (*e.g.*, *high school*) that are more often of interest to users.

4.7 Conclusions

This chapter explored what additional information can be learned to better understand and make use of the entity phrases that we find as Web extraction arguments. We began by designing an efficient mechanism to link millions of extraction arguments to their Wikipedia articles, as doing so can disambiguate them and provide their Freebase types. After finding that a significant fraction of Web noun phrases do not link to any entities in Wikipedia, we proposed the *unlinkable noun phrase problem* of detecting and typing unlinkable entities. We then designed initial solutions for this problem that significantly outperformed appropriate baselines. Last, we proposed a technique for splitting noun phrases that not only are not in Wikipedia, but are also ambiguous. We evaluated and verified the efficacy of our techniques using a set of 15 million ReVerb Open IE extractions. Our proposed techniques enable deeper understanding of the entity phrases and also enable numerous applications. The linking and typing components have also been run on 1 billion extractions, enabling demos such as the one shown in Figure 4.9.

Several avenues of improving this work include exploring additional features, incorporating additional resources, and jointly executing system components. For entity linking, there may exist additional useful *corpus-level features* that remain to be discovered. For detecting new entities, new features could be designed around detecting keywords that tend to occur more often with non-entities (*e.g.*, "different"), as well as around analyzing behavior over time of substrings of the full noun phrases. For predicting semantic types, we can incorporate additional signals such as shared term heads. As an example, if two terms both take the form "* river" or "* national park" then they are likely to share types. For many entity types this kind of signal is not available. However, in cases where it is available, it could aid in finding good similar entities. We can also explore the use of additional resources such as comprehensive alias lists [116] to help linking or timestamped Twitter data to help entity detection. Feeding back system output to different components might also improve system performance. For example, non-entity noun phrases that make it to the typing step might lead to particular predicted type distributions that indicate an error occurred earlier in the process.

Another future extension would be adapting the techniques presented in this chapter to *link relation phrases*. While there do not currently exist relation repositories with coverage comparable to Freebase and Wikipedia's coverage over entities [82], such a resource could exist in the future (*e.g.*, if Freebase expands its coverage of relations). As with entity linkers, we could structure a relation linker to also have a candidate generation step and then a candidate ranking step. Candidate generation could generate a list of candidates using any known textual forms of the relations (from systems such as [46]), string matches against the relation names, and compatibility of relations in terms of domain and range types. Candidate ranking could then score candidate relations based on features such as how closely the instances match known instances of the candidate relations (incorporating entity set expansion [86] as needed), string match level on relation names, the known prominence of the candidate relations, and context matching with known instances of the candidate relations. As with entity linking, we can also calculate a link score and an ambiguity score, and the combination of the two will help us judge whether a relation phrase is unlinkable.

Over the first several chapters, we have presented how "open" information extraction over Web text can extract large amounts of general textual information, and then how this extracted information can be cleaned and associated with semantic data for increased understanding and usability. We first analyze the full extractions, then the relation phrases, and then the entity phrases. For each, we show how incorporating large, general-coverage knowledge bases such as Freebase and Wikipedia can lead to better performance. At the same time, we are careful to ensure that incorporating the KBs does not restrict the vocabulary that can be processed, as that is a key advantage of open-domain Web text processing.

A driving promise of Web text processing is that it can enable compelling new capabilities. The techniques proposed thus far for leveraging KBs in Web text processing apply not just to Information Extraction, but also to other open-domain Web text processing tasks that may be even more directly user-facing, such as Web search. In the next chapter, we will introduce a novel and useful user-centric experience that we can enable when the techniques described in this chapter are applied to a Web search scenario.

Chapter 5

BEYOND EXTRACTION: ACTIONS FOR WEB SEARCH

The techniques presented in this thesis have applicability in areas beyond information extraction. Another popular area for open Web text processing is Web search. Web search must handle any terms and entities a user queries for, so many domain-independent techniques are used. Can knowledge bases also be effectively leveraged here?

Just as in entity linking of extractions, entity linking of search queries could match using factors such as string match, prominence and context match. With unlinkable entities in extractions if we could not link the noun phrase "Sushi Girl", then we would observe relations it occurs with (*e.g.*, "will be released in") and find linkable entities also appearing with those relations (*e.g.*, "The Dark Knight") to determine that it is a *film*. Similarly, if a user issues a Web search query for "Sushi Girl" and we cannot link it, then we can observe the context words that users have issued alongside it in queries (*e.g.*, "trailer", "director", "plot") as well as its top search result hosts (*e.g.*, "imdb.com" and "moviefone.com") and find linkable entities which also appear with those query context words and search results (*e.g.*, "Avatar") to propagate the *film* type.

In extraction, linking and typing entities enables new experiences such as typed question answering. We propose that in search, linking and typing entities can also enable compelling new experiences. This chapter introduces a Web search experience called *Actions for Web search* that we developed on top of the linking and typing capability. The idea is that when users issue search queries (*e.g.*, "The Dark Knight") there are often specific *actions* they wish to accomplish (*e.g.*, "Read Reviews" or "Watch Trailer"). If search engines can identify and type the entities within queries, then they can also automatically predict and broker these actions. In the context of Web search data, actions are *implicit* information that need to be inferred. KBs can help us better understand and apply not only the diverse information expressed explicitly in Web text (*e.g.*, relations and arguments), but

also information expressed implicitly in Web text (*e.g.*, actions). Contributions of this chapter include:

1. Conceptual: We introduce the *Actions for Web search* paradigm. We establish that there are specific Web actions for users to perform on entities. We conduct an annotation study on search query logs which empirically verifies that a large proportion of sampled query-click pairs reflect actions on entities.

2. Modeling: We propose probabilistic models to generate entity bearing queries from actions, incorporating information from context words, clicked hosts, and entity types.

3. Implementation: We train our models on three months of query data from a commercial search engine, and address the necessary end-to-end system issues for producing a system to recommend suitable actions for new queries.

4. Experimental: We conduct a user-study to evaluate our different models, showing which model components are most important for generating actions.

5.1 Actions for Web Search

Entities are central to a large fraction of Web search queries. Whether users seek to find information about an entity or transact on the entity (*e.g.*, "[buy] toy story 3," "[watch or listen to] obama weekly address"), understanding the underlying query intent is key to providing a rich search experience.

Web search today has already taken great strides away from simple query word matching. For example, head entities in large query segments (*e.g.*, local, entertainment, shopping) are routinely recognized in queries and rich direct displays are presented to users by filling editorially-defined templates with associated structured data. For example, a query for "lion king" on Bing yields such a direct display consisting of an image of the movie cover, showtimes at local theaters, the running time, genre, and ratings of the movie. However, since the focus is on the *dominant* actions, the search engine underserves, for instance, a Netflix user seeking other actions such as adding the movie to her streaming queue, or a

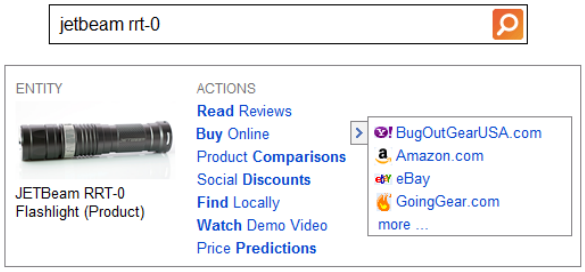

Figure 5.1: **Search as an Action Broker:** In the future, search engines could directly broker the actions that users want to take.

child trying to find a toy figurine. In addition, a different movie such as Michael Moore's most recent documentary would certainly have a different underlying intent distribution. Also, actions associated with queries for tail entities such as flashlights or small vineyards are completely ignored.

Search as an action broker: A promising future search scenario involves modeling the user intents underlying the queries and brokering the Web pages that accomplish the intended actions. In this vision, the broker is aware of the entities and actions of interest to its users, understands the intent of the user, ranks all providers of actions, and provides direct actionable results through APIs with the providers. For example, consider a user who queries for "jetbeam rrt-0," a flashlight. The broker would recognize the particular entity mentioned in the query, and would return a personalized ranked list of actions to the user. Figure 5.1 provides a simplistic illustration of how this user experience could look on a search results page. With actions present, users could save clicks and save time, and sometimes even discover new actions to help them toward their goals. New revenue streams open up from paid action placement, lead generation, and on-site commercial transactions.

This chapter addresses several key questions that arise within this proposed paradigm. Do most Web queries lend themselves to actions on entities? What does the space of actions look like? And most importantly, given a query with an entity (*e.g.*, identified via techniques analogous to those in Chapter 4), how can a search engine determine actions to recommend?

We begin with an annotation study conducted over query-click logs from Bing to deter-

mine what fraction of queries contain entities, and whether these queries tend to map to particular actions that can be performed on entities. We then motivate and design generative models, the most complex of which accounts for queries, entities, actions, textual query contexts, entity types, and historical click data.

An automated approach to learning relevant actions for queries is necessary because there are too many possible distinct Web queries for editors to manually pair with actions. Manually preparing top actions for each entity type would also be insufficient because it would not account for context words in queries (*e.g.*, the queries "Microsoft jobs" and "Microsoft software" should lead to different actions despite sharing the same entity), entities of the same type can have different top actions (*e.g.*, queries for a 2012 Ferrari may historically lead to clicks on topcarwallpapers.com while queries for a 1995 Ford historically lead to clicks on a used car value site), and top actions for an entity may change over time. For example, a query for the next iPhone would have different desired actions a year before launch, a week before launch, at launch, and a month after launch. The use of automated methods enables frequent re-training through a more recent data set of query-click logs.

In addition to presenting the models, we also explore a number of issues that need to be addressed in going from a theoretical generative model of actions to an actual end-to-end search engine component that is able to recommend appropriate actions when given a new query. Finally, we conclude with a user study evaluating the performance of our models.

5.2 Related Work

Beyond the ideas of identifying and typing entities, related work that we build upon includes entity-centric search and intents. Work that we differentiate ourselves from includes previous work on actions and topic modeling using query logs.

5.2.1 Entity-Centric Search

As proposed in Dalvi *et al.* [26], Entity-Centric search focuses on creating a "semantically rich aggregate view of all the information available on the Web for each concept instance." Researchers have typically focused on techniques for automatic generation of topic pages based on entities (*e.g.*, [3, 105]), or on tailored information for particular entity classes (*e.g.*,

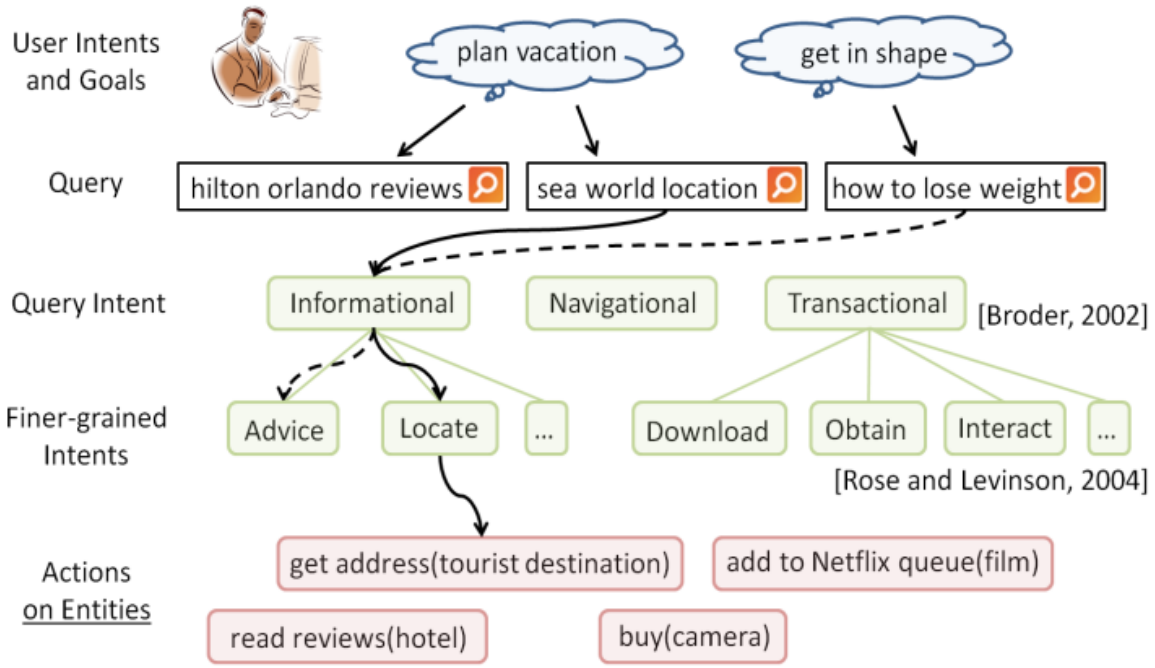

Figure 5.2: **Actions Differ from Intents:** Actions must be performed on Entities, and are often more specific and grounded.

popular search engines displaying sports scores when given a sports team query). To the best of our knowledge we are the first to propose learning and presenting specific sets of *actions* for each entity.

5.2.2 Intents

Queries can be associated with higher-level Intents such as "planning a vacation" or "getting in shape" (see Figure 5.2). Broder [13] outlined three main intents: Informational, Navigational, and Transactional. Rose and Levinson [102] further divided informational and navigational intents into subcategories resulting in 11 finer-grained intents. Yin and Shah [131] used search logs to organize taxonomies of intent phrases, and Jansen *et al.* [51] studied how to classify queries into intents.

Our notion of *actions* is at an even finer level. Actions are very specific versions of intents that are performed on entities. Some actions overlap with finer-grained intents

(*e.g.*, "download"), but the majority of intents (*e.g.*, "interact") are not concrete enough to be suggested to users. While some queries map easily into both intents and actions (*e.g.*, "sea world location" in Figure 5.2), there are also queries that have a clear intent but do not contain any entity and hence cannot be associated with an action (*e.g.*, "how to lose weight"). Task intents covering multiple actions (*e.g.*, "book trip") are also out of scope.

5.2.3 Actions

Actions and action ontologies have been previously explored in robotics, intelligent agents, and philosophy (*e.g.*, [55, 75]), but the primary focus in those areas was to develop a standardized set of actions (with pre-conditions and post-conditions) that could guide the planning processes of intelligent agents. In contrast, when we refer to an *action for Web search*, we refer to actions for human users to perform over the Web. Most of these actions (*e.g.*, "read reviews" or "download") have no important prerequisites, while for those that do (*e.g.*, "add to Netflix queue" requires a Netflix account), we assume that the preconditions can be addressed based on information known about the users.

5.2.4 Use of Probability Models

There has been prior work in using probability models for modeling user queries. For example, Carman *et al.* [19] extended Latent Dirichlet Allocation (LDA) [10] to rank documents by likelihood of the model given a particular query and user pair. Their model accounted for users, clicked documents, and query terms. Gao *et al.* [39] adapted statistical machine translation techniques to learn how document titles are semantic translations of queries. Guo *et al.* [41] used probability models to identify named entities and entity classes from query logs. Our work differs in that the primary focus of our models is on learning actions, a variable which other studies have not modeled.

5.3 Annotation Study

To confirm the potential utility of providing *actions*, we begin with a manual study of entities and actions in Web searches. We collect a frequency-weighted query sample of 200

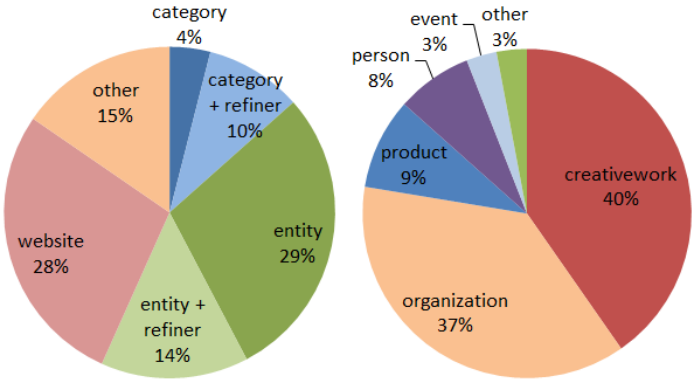

Figure 5.3: **Distribution of Entity Types in Web Search:** *Left*: At 200 labels, 43% of the queries contained entities, and 14% contained entity categories. *Right*: Distribution of entities into Schema.org types.

query-click pairs. We examine each query to determine whether it contains an entity and whether we can infer an action that the user intends to accomplish given the query and the clicked host. Although one can only observe trends on such a small sample set, these results will serve as a guide for our automatic action induction models described in Section 5.4.

Throughout this chapter, we define an action as follows:

Action: An empirically observable, direct manipulation or information request on an entity.

We target actions that are useful in the context of Web search. For example, "interact" is too coarse and "drink" is not an action that can be accomplished on the Web. Examples of useful actions are: "buy," "add to movie queue," and "read reviews."

5.3.1 Entities in Queries

We divided queries into four groups with respect to the presence of an entity in the query: (i) contains an entity; (ii) contains an entity category (*e.g.*, "car battery"); (iii) contains a website entity (URL or website name); and (iv) all other queries. Figure 5.3 summarizes the frequency of each group, further separating out whether a query contains refiner words (*e.g.*, "download GoldenEye" with the refiner "download") in addition to the entity or entity

category.

43% of the queries contain an entity (29% by itself, 14% with a refiner), 14% contain an entity category (4% by itself, 10% with a refiner), 28% contain a reference to a website, and 15% do not contain any entity. Website references often occur in navigational queries where the user intends to visit a particular site, which leaves 57% of queries (43% + 14%) that have entities or entity categories. None of our annotated queries contain multiple entities. Guo *et al.* [41] found that 71% of search queries contained named entities, although they neither specify whether they consider frequency of individual queries, nor how they classify entity categories and website entities. Summing our entity, entity + refiner, and website entity categories, we end up with a proportion of entities in queries matching their results.

Next, we examined the types (or taxonomy categories) of the entities that we found. For entity types, we refer to the top level of the Schema.org entity taxonomy, which is a collection of schemas developed jointly by Bing, Google and Yahoo, designed explicitly with the intent of facilitating Web search over entities on the Web.

Within our sample of entities, we found that the most popular Schema.org top-level category was *CreativeWork* at 40%. This is a fairly broad category that covers all books, movies, songs, software, *etc.* The category *Organization* covered 37% of our entity sample. The *Organization* type covers hotels, restaurants, government organizations, local businesses, *etc.* There was also the occasional *Product* at 9%, and *Person* type at 8%. *Event* type covered 3% and the last 3% fell into other various types.

5.3.2 Actions in Queries

Next, we examined how often the queries in our sample can be associated with specific actions on entities. We also estimate whether the actions in Web search are enumerable.

We manually inferred the actions that are associated with each sample query by examining the raw query strings (consisting of entities and possible refiner words), and the clicked URLs. In the majority of cases, this information clearly indicated a particular action (*e.g.*, "yahoo messenger download" clearly indicates the action "download"). In the absence of refiners in the query, the clicked URL generally gives a good signal to identify the action. For

example, a query for "Hobart corporation" with a click on "http://hobartcorp.com/Contact-Us/" indicates the intended action "get contact information." 19 of the query/URL pairs in our sample were ambiguous with respect to the intended action, *e.g.* "GEICO insurance"-"www.geico.com", where the specific intended action is not clear. In some of these cases we took likely potential actions from inspecting the clicked sites (*e.g.*, "see menu" on a restaurant URL) and added them to our inventory of actions.

From our 200 queries, we compiled a list of 47 actions. Some of the most popular actions included "login," "play game," "read news about," and "shop for." Some of the less common actions included "find recipe for," "find lyrics," and "get hours of." Working through the 200 queries, our discovery rate of new actions dropped from over 20 distinct actions for the first 50 labels to fewer than 5 new actions for the last 50 queries. This suggests that there is an enumerable primary set of actions that users perform in the context of Web search.

5.4 Action Induction

We turn our attention now to the tasks of automatically learning the underlying actions intended in Web search as well as to recommending actions given new queries. Our approach is to probabilistically describe how *actionable queries*, *i.e.*, queries containing an entity and underlying action intent, are generated by Web search users. The models we develop theorize that user query sessions are governed by latent actions and entity types, which influence the choice of query terms and clicks. Inference procedures are then used to infer actions after learning these models by maximizing the probability of observing a large collection of real-world queries and their clicked hosts.

In this section we initially present two graphical models (summarized in Figure 5.4). To generate queries from actions, our Model 1 models query context and clicked URLs. Model 2 builds on Model 1 by also modeling entity types, and explicitly observing entities. Then, we propose an extension to each model that adds a switch variable to better handle queries with empty contexts.

Each query q is represented by a triple $\{n_1, e, n_2\}$, where e represents the entity mentioned in the query, n_1 and n_2 are respectively the pre- and post-entity contexts (possibly empty), referred to as refiners. As a running example, we consider a user who is interested in

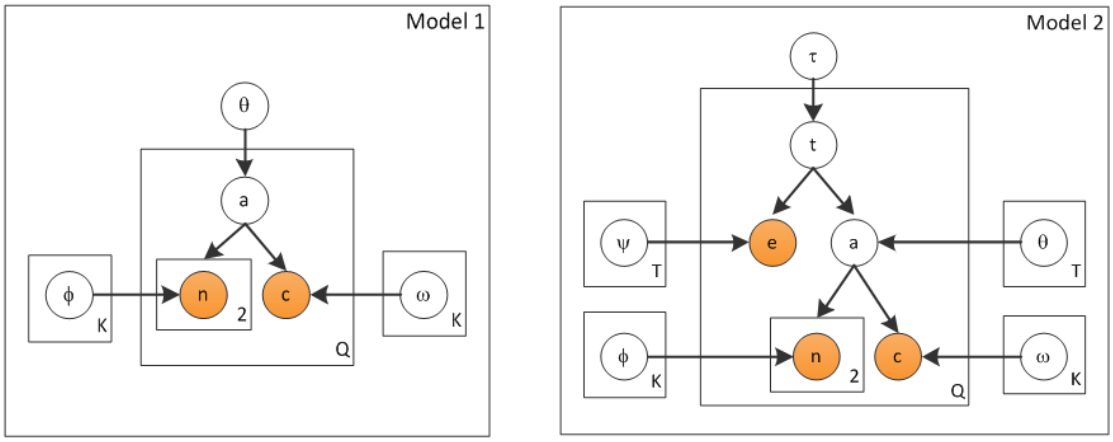

Figure 5.4: **Generative Models for Actionable Queries:** Model 1 includes query context words n and host clicks c, and Model 2 adds the entity type t and the entity e. Shaded circles are observed variables.

reading a review about the movie "Inception," and who issues the query "inception review" to a search engine. Here $n_1 = \emptyset$, $e =$ "inception," and $n_2 =$ "review." Details on how we obtain our corpus are presented in Section 5.5.

5.4.1 Model 1 (context+click)

The choice of refiner words in a query is clearly influenced by the intended action. For example, words such as "review," "ebert," and "opinion" are more likely to be used in a query if the intent is to read a review. Host clicks are also correlated with action intents. For example, clicks on "rottentomatoes.com," "epinions.com," and "dpreview.com" are more likely if the user has the intent to read reviews, whereas clicks on "bestbuy.com" and "ebay.com" are more likely for a buying intent. Broder *et al.* [13] also found hosts associated with queries to be useful in classifying queries.

Our first probabilistic graphical model, Model 1, leverages these signals. It generates actionable queries by first picking an action from a distribution over a set of latent actions, then choosing query context words n_1 and n_2, and then clicking on a host c. This model does not explicitly capture the entity in the query, and hence a query is represented by the pair $\{n1, n2\}$. The generative process below summarizes the model illustrated on the left

in Figure 5.4:

Model 1: Generative model of actionable queries.

For each query q

action $a \sim$ Multinomial(θ)

l-context $n_1 \sim$ Multinomial(ϕ_a)

r-context $n_2 \sim$ Multinomial(ϕ_a)

click $c \sim$ Multinomial(ω_a)

In our running example for the query "inception review," our model first generates the action "read reviews," then given this action chooses the refiner words $\emptyset$ and "review," and then generates a click on a site such as "rottentomatoes.com."

The joint probability of the model is the product of the conditional distributions, as given by:

$$P(a, q{=}\{n_1, n_2\}, c \mid \theta, \Phi, \Omega) = P(a \mid \theta)P(n_1 \mid a, \Phi)P(n_2 \mid a, \Phi)P(c \mid a, \Omega) \tag{5.1}$$

Next, we define each of the terms in the joint distribution. Let K be the number of latent actions that govern our query log, where K is fixed in advance. Then, the probability of action a is defined as a multinomial distribution with probability vector θ, such that the probability of a particular action is given by:

$$P(a{=}\hat{a}) = \prod_{k=1}^{K} \theta_k^{I[k=\hat{a}]} \text{ , s.t. } \sum_k \theta_k = 1 \tag{5.2}$$

where I is an indicator function set to 1 if its condition holds, and 0 otherwise.

Let V be the shared vocabulary size of all query refiner words n_1 and n_2. Given an action a, the probability of generating a refiner n is given by a multinomial distribution with probability vector ϕ_a such that $\Phi = [\ \phi_1, ..., \phi_K]$ represents parameters across actions:

$$P(n{=}\hat{n} \mid a{=}\hat{a}) = \prod_{v=1}^{V} \Phi_{\hat{a},v}^{I[v=\hat{n}]} \text{ , s.t. } \forall a \ \sum_{v=1}^{V} \Phi_{a,v} = 1 \tag{5.3}$$

Finally, we assume there are H possible click values, corresponding to H Web hosts. A click on a host is determined by an action. Given an action a, we assume the probability of generating a click on host c is a multinomial with a probability vector ω_a such that $\Omega = [\ \omega_1, ..., \omega_K]$ captures the matrix of parameters across all K actions. In particular:

$$P(c{=}\hat{c} \mid a{=}\hat{a}) = \prod_{h=1}^{H} \Omega_{\hat{a},h}^{I[h=\hat{c}]} \text{ , s.t. } \forall a \sum_{h=1}^{H} \Omega_{a,h} = 1 \tag{5.4}$$

Inference: Given a query, we apply Bayes' rule to find the posterior distribution over the actions. In particular, the posterior distribution, $P(a|q,c)$, is directly proportional to the joint distribution. We can exactly compute this distribution by evaluating the joint for every value of a and the observed configuration of q and c.

Learning: Given a query corpus Q consisting of N independently and identically distributed queries (each $q^j = \{{n_1}^j, {n_2}^j\}$) and their corresponding clicked hosts, we estimate the parameters Θ, Φ and Ω that maximize the (log) probability of observing Q. The log $P(Q)$ can be written as:

$$\log P(\mathcal{Q}) = \sum_{j=1}^{N} \sum_{a} P^j(a \mid q,c) \log P^j(q,c,a) \tag{5.5}$$

In the above equation, $P^j(a|q,c)$ is the posterior distribution over actions for the j^{th} query. We use the Expectation-Maximization (EM) algorithm to set the parameters. Starting with a random initialization of the parameters, EM iterates between the E-step in which $P^j(a|q,c)$ is computed for each query (assuming parameters are fixed as computed in the previous M-step) and the M-step in which the parameters are updated by fixing $P^j(a|q,c)$ to the values computed in the E-step.

The parameter updates are obtained by computing the derivative of log $P(Q)$ with respect to each parameter, and setting the resultant to 0. The update for θ is given by the average of the posterior distributions over the actions:

$$\theta_{\hat{a}} = \frac{\sum_{j=1}^{N} P^j(a{=}\hat{a} \mid q, c)}{\sum_{j=1}^{N} \sum_a P^j(a \mid q, c)} \tag{5.6}$$

For a fixed a, the update for ϕ_a is given by the weighted average of the context words, where the weights are the posterior distributions over the actions, for each query. In particular:

$$\Phi_{\hat{a},\hat{n}} = \frac{\sum_{j=1}^{N} P^j(a{=}\hat{a} \mid q, c)\Big[I[n_1^j{=}\hat{n}] + I[n_2^j{=}\hat{n}]\Big]}{2\sum_{j=1}^{N} P^j(a{=}\hat{a} \mid q, c)} \tag{5.7}$$

Similarly, we can update Ω, the parameters that govern the distribution over clicked hosts for each action. For a fixed a, it is updated by taking the weighted average of the clicked hosts, with weights provided by the posterior distribution over the actions:

$$\Omega_{\hat{a},\hat{c}} = \frac{\sum_{j=1}^{N} P^j(a{=}\hat{a} \mid q, c) I[c^j{=}\hat{c}]}{\sum_{j=1}^{N} P^j(a{=}\hat{a} \mid q, c)} \tag{5.8}$$

5.4.2 Model 2 (context + click + type + entity)

The semantic type of the entity mentioned in the query is often strongly correlated with the intended action. For example, if the queried entity is a movie, the user is likely to be looking to buy it, rent it, view local showtimes, or buy theater tickets. It is unlikely however that the user is interested in hacking it, getting its address, or connecting to it. Similarly, a "read biography" action is more likely for a *person* entity and a "view stock price" action is more likely for a *corporation* entity. By accounting for types, the model can avoid recommending incorrect typed actions, such as "view stock price" on a person entity.

In addition, entities themselves are usually instances of very few types and hence we expect them to be helpful in disambiguating the types. Therefore, in this model, we explicitly model the entities and their types. The right side diagram of Figure 5.4 illustrates the graphical model. The generative process for Model 2 is as follows:

Model 2: Generative model of actionable queries.

For each query q

type $t \sim$ Multinomial(τ)

action $a \sim$ Multinomial(θ_t)

entity $e \sim$ Multinomial(ψ_t)

l-context $n_1 \sim$ Multinomial(ϕ_a)

r-context $n_2 \sim$ Multinomial(ϕ_a)

click $c \sim$ Multinomial(ω_a)

Note that in our generative model, we are assuming that the action is generated independently of the entity itself. However, the choice of the entity also influences the subset of actions that are possible for a particular choice of the type. The independence assumption between actions and entities is a matter of mathematical convenience. Otherwise, we require learning a parameter for each action-type-entity configuration, giving rise to a huge number of parameters. Instead, we choose to include these dependencies at the time of inference, as described later.

For our running example, Model 2 first generates the type "film," then given the type, it generates the entity "inception," and then generates the action "read reviews." The action is used to generate the pre- and post- context words $\emptyset$ and "review," and then the click on a site such as "rottentomatoes.com."

The joint probability over the model variables is:

$$\begin{aligned} P(t, a, q{=}\{n_1, e, n_2\}, c \mid \Theta, \Phi, \Omega, \tau, \Psi) = & P(t \mid \tau) P(a \mid t, \Theta) P(e \mid t, \Psi) \\ & P(c \mid a, \Omega) P(n_1 \mid a, \Phi) P(n_2 \mid a, \Phi) \end{aligned} \tag{5.9}$$

Next, we describe each term in the joint probability. Let T be the number of entity types. The probability of generating a type t is governed by a multinomial with probability vector τ. In particular:

$$P(t{=}\hat{t}) = \prod_{i=1}^{T} \tau_i^{I[i=\hat{t}]} \text{ , s.t. } \sum_{i=1}^{T} \tau_i = 1 \tag{5.10}$$

Let E be the number of known entities. The probability of generating an entity e given type t is a multinomial with a probability vector ψ_t such that $\Psi = [\ \psi_1, ..., \psi_T]$ captures the matrix of parameters across all T types. In particular:

$$P(e{=}\hat{e} \mid t{=}\hat{t}) = \prod_{i=1}^{E} \Psi_{\hat{t},i}^{I[i=\hat{e}]} \text{ , s.t. } \forall t \sum_{i=1}^{E} \Psi_{t,i} = 1 \tag{5.11}$$

Since actions are now conditioned on types, for every value of type, it is a multinomial distribution with probability vector θ_t such that $\Theta = [\ \theta_1, ..., \theta_T\ ]$ represents parameters across types:

$$P(a{=}\hat{a} \mid t{=}\hat{t}) = \prod_{k=1}^{K} \Theta_{\hat{t},k}^{I[k=\hat{a}]} \text{ , s.t. } \forall t \sum_{k=1}^{K} \Theta_{t,k} = 1 \tag{5.12}$$

Prior distributions over the context words and clicked host remain unchanged as in Model 1.

Inference: Given a query, and the learned model, we can apply Bayes' rule to find the posterior distribution, $P(a, t|q, c)$, over the actions, as it is proportional to $P(a, t, q, c)$. We compute this quantity exactly by evaluating the joint for each combination of a and t, and the observed values of q and c.

During inference, we also enforce that for an entity, there are only certain admissible types. As an example, if the entity is *Inception*, valid types include *film* and *book*. We set the posterior probability of invalid types (and hence the relevant type-action configurations) to zero. We obtain the set of admissible types for every entity using an external knowledge base. In this chapter, we use Freebase (see Section 5.5.1). A desirable side effect of this strategy is that only valid ambiguities are captured in the posterior distribution. Thus the model can focus on capturing the actions for multiple of its *valid possible* senses (types).

Learning: As in the previous model, we perform maximum likelihood estimation of the parameters using the EM algorithm. Below, we present M-step update equations for some of the parameters that are unique to this model. Other parameter updates are similar in spirit to Model 1.

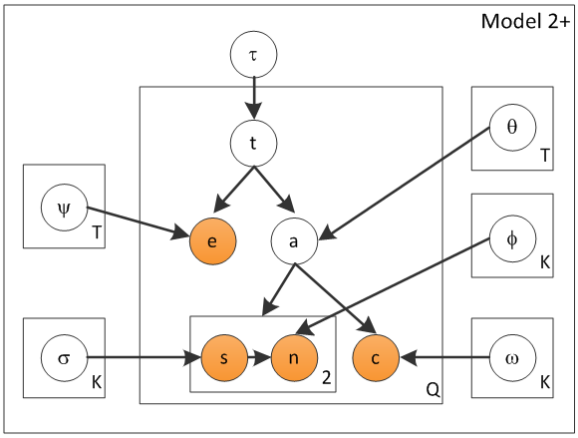

Figure 5.5: **Generative Model with Empty Context Switch:** Model 2^+ adds an empty context switch s. Shaded circles are observed variables.

$$\tau_{\hat{t}} = \frac{\sum_{j=1}^{N} \sum_{a} P^j(a, t{=}\hat{t} \mid q, c)}{\sum_{j=1}^{N} \sum_{a,t} P^j(a, t \mid q, c)} \tag{5.13}$$

$$\Psi_{\hat{t},\hat{e}} = \frac{\sum_{j=1}^{N} \sum_{a} P^j(a, t{=}\hat{t} \mid q, c) I[e^j{=}\hat{e}]}{\sum_{j=1}^{N} \sum_{a} P^j(a, t{=}\hat{t} \mid q, c)} \tag{5.14}$$

5.4.3 Empty Contexts

Generally in Web search, most query contexts are left empty. For example, users tend to issue the query "obama" far more frequently than queries with refiners such as "support obama" or "obama schedule." In fact, upon inspection of the Φ table for Models 1-2, we noticed that over 90% of the probability mass is covered by the empty context. In order to spread that mass to useful context words, we explicitly represent the empty context using a switch variable that determines whether a context will be empty. Figure 5.5 illustrates how we model the switch in Model 2, creating Model 2^+. The generative story for both Models 1 and 2 can be augmented as follows:

Model X + Switch:

For each query q

...

~~l-context $n_1 \sim$ Multinomial(ϕ_a)~~

~~r-context $n_2 \sim$ Multinomial(ϕ_a)~~

switch $s_1 \sim$ Multinomial(σ_a)

switch $s_2 \sim$ Multinomial(σ_a)

if (s_1) l-context $n_1 \sim$ Multinomial(ϕ_a)

if (s_2) r-context $n_2 \sim$ Multinomial(ϕ_a)

...

Incorporating the switch into the joint probability of each model is straightforward. Below we show it for Model 2:

$$\begin{aligned} P(t, a, q{=}\{n_1, e, n_2\}, c, s{=}\{s_1, s_2\} \mid \Theta, \Phi, \Omega, \tau, \Psi, \sigma) = \\ P(t \mid \tau) P(a \mid t, \Theta) P(e \mid t, \Psi) P(c \mid a, \Omega) \\ \prod_{i=1}^{2} P(n_i \mid a, \Phi)^{I[s_i=1]} P(s_i | a, \sigma) \end{aligned} \tag{5.15}$$

The probability of generating an empty or non-empty context s given action a is given by a Bernoulli with parameter σ_a:

$$P(s \mid a{=}\hat{a}) = \sigma_{\hat{a}}^{I[s=1]} (1 - \sigma_{\hat{a}})^{I[s=0]} \tag{5.16}$$

The M-step update function for the switch parameter σ is:

$$\sigma_{\hat{a}} = \frac{\sum_{j=1}^{N} \sum_t P^j(a{=}\hat{a}, t \mid q, c, s) \Big[I[s_1{=}1] + I[s_2{=}1] \Big]}{2 \sum_{j=1}^{N} \sum_t P^j(a{=}\hat{a}, t \mid q, c, s)} \tag{5.17}$$

In the above models, we learned point estimates for the parameters (Φ, Θ, Ω, τ, Ψ, σ), that govern the variables of interest, including type, actions, context, entities and clicks. One

can take a Bayesian approach and treat these parameters as variables (for instance, with Dirichlet and Beta prior distributions), and perform Bayesian inference. However, exact inference will become intractable and we would need to resort to methods such as variational inference or sampling. We found this extension unnecessary, as we had a sufficient amount of training data to estimate all the parameters well. In addition, our approach enabled us to learn (and perform inference in) the model with large amounts of data with reasonable computing time.

5.4.4 Enforcing Action Diversity in Learning

In training Model 2 using the EM algorithm, we found that the local optimal solutions often amounted to action clusters that were tied very strongly to specific types. For instance, the *athlete* entity type had a $P(Action|Type)$ of 95% into an action cluster that focuses on sports statistics. While it is desirable that the model learns a good top-ranked action (*e.g.*, "Retrieve Sports Statistics"), we also want to be able to recommend a full range of actions for queries (*e.g.*, for the *athlete* type we would also want to see the next top actions, such as "Follow on Social Networks," "Read Biography," "View Pictures" and "Buy Tickets to see"). If one top action absorbs too much probability mass, we often observe empirically that the lower-ranked actions do not gain sufficient probability mass. This is clearly an artifact of the EM algorithm-based learning paradigm.

We resolve this through a two-step procedure for learning. In the first step, we run EM iterations to learn only the parameters that do not involve the entity type (*i.e.*, by freezing the Θ parameter). This allows Model 2 to learn action clusters tied more closely to query contexts and clicked hosts. In a second step, we continue learning with additional EM iterations, now also letting the algorithm learn the Θ parameter. We found that this strategy reduces the average amount of mass for the top-ranking action clusters, which in turn leads to probability mass being more evenly distributed across actions and ultimately to better ranking of the action clusters. In one experiment, we found that this two-step learning reduced the average top $P(Action|Type)$ value from 48% to 28%, distributing the mass more evenly across other actions.

5.4.5 Decoding

Consider a runtime scenario where a new search query $q =$ *"new york city hotels"* is received. Decoding is accomplished as follows. First, we identify the entity $e =$ *"new york city"*. This leaves the query contexts as $n_1 = \emptyset$ and $n_2 =$ *"hotels"* (and switch values $s_1 = true$ and $s_2 = false$). We use historical search query data to identify a distribution $P(c|q)$ over all hosts $c \in H$ that received a click for this query in the past. The recommendation score (probability) of an action a is then:

$$P(a \mid q{=}\{n_1, e, n_2\}, c, s) = \sum_t \sum_{c \in H} P(a, t \mid q, c, s) P(c \mid q) \tag{5.18}$$

The parameter Ω can be directly looked up to rank hosts given each action a. Note that if no click history is available, for instance if observing a query with a never before seen entity, the model can still recommend actions using its other parameters. Also, if the candidate types of an ambiguous entity are known, then we can return an action distribution given each type. If the types are unknown, then we can return an action distribution over each latent type. In both cases, we can marginalize the types to get an action distribution for the query.

5.4.6 Cluster Labeling: Web Action Phrases

The action clusters discovered by our models are clusters of words defined by the Φ parameter. We need to "translate" each action into action recommendation phrases that can be presented to the user (*e.g.*, "read reviews" or "download").

We begin by examining the most probable context words for each action. The leftmost word cloud in Figure 5.6 illustrates this for one of our discovered actions (Appendix E shows more examples). Clearly the Figure 5.6 cluster relates to downloading free software.[1] We then tease out the "actions" by obtaining a list of verbs/action words, and then intersecting this list against the context words in the clusters.

[1]Note that '@' is a wildcard for any digit. Thus "@.@" is a placeholder for software versions such as "3.1" or "2.0."

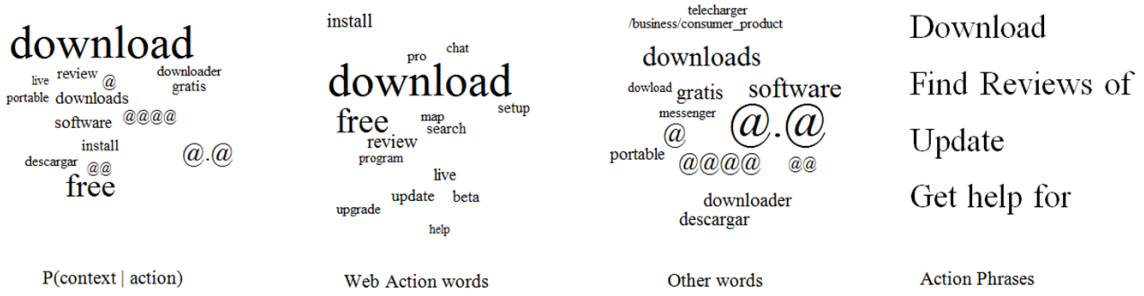

Figure 5.6: **Web Action Words:** To obtain Action Phrases we first identify top Web Action words from the action's most likely context words.

Using a generic verb list is not ideal here because we are restricted to actions that users can perform on the Web, many verbs do not take people in the agent role (*e.g.*, "merge"), and generic verb lists often do not contain words that can be used as Web-based actions such as "blog," "podcast," or "torrent." To obtain a list of appropriate actions, we defined a few key lexical patterns (similar to Hearst [43]) that generally contain action words, such as *"want to (x)"*, *"have to (x)"*, *"you can (x)"* and *"I can (x)"*.

We then obtain the most frequent instances of *(x)* by applying these patterns against a large Web body-text trigram corpus. After filtering out adverbs (using 21 additional patterns, designed to catch adverbs in this corpus) and filtering out noise (the 25% of actions with the lowest frequency / unigram count, *e.g.*, "a" and "boy"), this leaves us with a list of 13,417 action words. This list still contains a number of actions (*e.g.*, "shock" or "kill") that users cannot perform over the Web, so we filtered it down to the 1,279 *Web actions* that also occurred with the pattern "(x) at (y)" in our trigrams, where (y) takes the form of a website URL (*e.g.*, "Amazon.com"). Examples of the most popular Web actions include: "buy," "review," "shop" and "unsubscribe."

The second word cloud in Figure 5.6 illustrates $P(n|a)$ for those contexts n that passed our filter. The third word cloud shows the remaining words when Web action words are removed. The resulting three word cloud types, illustrated in Figure 5.6, are used as a tool for a human-annotation task to specify the appropriate action phrases for each cluster. From our automatically generated word clouds of action words, non-action words, and the

website	product line	digital camera
consumer product	software	film
computer/video game	person	athlete
politician	actor	artist
employer	business operation	restaurant
location	travel destination	tourist attraction
sports facility	university	road

Table 5.1: **Actions Study Entity Types:** The actions study uses 21 initial Freebase types, which were chosen based on the earlier annotation study.

popular hosts for each action cluster, we found it easy for annotators to specify these action phrases. In future work we will explore techniques for fully automating this process of learning action phrases from action words.

5.5 Experimental Results

5.5.1 Data

We collected several months of queries issued to Bing and filtered them to retain only those that contain a signal for learning actions, by (i) removing any query that did not lead to a click and (ii) removing any query that did not contain an entity.

We cover a large number of oft-queried entities by focusing on the most important entity types discovered in our query analysis from Section 5.3 (see Figure 5.3). Note that Schema.org does not provide actual instances for their entity taxonomy, so we rely instead on Freebase for instances. For our initial experiments we chose types from Freebase that correspond to the most often queried types in Schema.org such as films, business operations, product lines, and people. Since Freebase is a fine-grained knowledge base, we also included subtypes such as athletes, actors and politicians, for a total of 21 total types (Table 5.1). The resulting sets account for approximately 3.4 million entity instances after de-duplication. In later work [87] we expand to more types, with an eventual goal of all types (as in Chapter 4).

As covered in Chapter 4, entity recognition across all data is a challenging problem and not always accurate. At model application time we need high quality entity recognition and entity to type mappings. For our model training however, given the large amount of available queries, we require only high precision entity recognition, so we turn to the following simple but effective method. We start by matching our query log with all our Freebase entity instances. To avoid problems like a query for "nice pants" getting matched to the city "Nice" in France, we apply an ambiguity filter on the capitalization ratio of our instances and allow matches on only the entities that appear capitalized at least 50% of the time in Wikipedia. To ensure that we do not match on substrings within entities (*e.g.*, if "Harry Potter" is the correct entity but not in our database of entities, we do not want to match on "Harry" or "Potter" separately), we also apply a standalone score filter [50] at 0.9, which calculates how often a string occurs as an exact match in queries relative to how often it occurs as a partial match.

For query contexts n_1 and n_2 defined in Section 5.4, although one could potentially use arbitrary ngram context sizes, we keep only queries where the contexts are empty or consist of single words (accounting for a very large fraction of the queries).

We define a navigational query as one where the user only wants to navigate to a specific site and is unlikely to be interested in any other action presented to her. We automatically eliminate such queries from the training set, where a query is considered navigational if in our logs it is associated with >1,000 clicks where >98% of clicks were to the same host (∼2% of our data points). Finally, we eliminate entries with clicked hosts that have been clicked fewer than 100 times over our entire query log.

After applying the filters described above, this yielded several million data points for training our models. Our data covers 235K distinct Freebase entities, 129K distinct context words, and 58K distinct click hosts. We refer to the resulting queries as *actionable queries* and denote the query set as Q according to Section 5.4.

5.5.2 Model Settings

We trained our models with 50 action clusters, set according to our earlier annotation study in Section 5.3.2, which found that this would give us good coverage over the main actions in Web search. Alternatively, the constraint could be alleviated by analyzing the semantic similarity between context words in the resulting clusters, or by using techniques similar to those for finding the optimal k in k-means [42], or by other methods such as those discussed by Blei *et al.* [10]. We conducted our two-step learning over 100 total EM iterations, running 2 folds per model.

5.5.3 Experimental Configurations

We used three test sets for our study:

- **HEAD:** 100 queries from a frequency-weighted random query sample of Q.
- **TAIL:** 100 queries from a uniform random sample of Q.
- **Type-Balanced:** 16 queries obtained as follows: Sampling starts from a frequency-weighted sample of Q, but during sampling, we only admit new queries to the test set if they cover a type that has not been covered yet.

The HEAD sample was used to test expected user impact in a Web search scenario whereas the TAIL sample tests how our method applies to rare entities. Whereas manually curated models could potentially address a large portion of head queries, only an automated method can model the tail. In our TAIL sample, we noticed that the entities were skewed towards the *person* type. We introduced the Type-Balanced set to test our model performance over a broad set of entity types, including less common types such as *university* and *tourist attraction*.

Finally, we report our results against the following models:

- **Baseline:** Simpler version of Model 1 that uses only query context words as observed variables, illustrated in Figure 5.7.

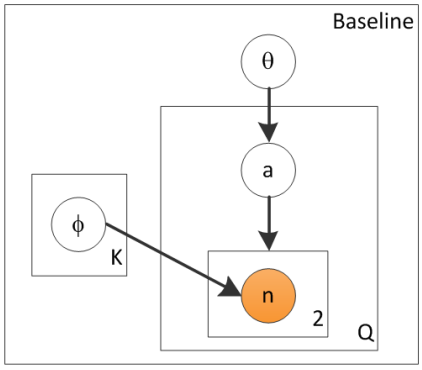

Figure 5.7: **Baseline Action Model:** A simple baseline using only context words.

- **Models 1, 2:** As described in Section 5.4.

- **Model 2^{+}:** Model 2 with the Empty Switch as described in Section 5.4.3 and illustrated in Figure 5.5.

There are 12 resulting experimental configurations.

5.5.4 User Study

We conducted a user study for each experimental configuration to determine relative effectiveness at discovering and suggesting actions. The goals of the study are to assess the following:

- End-to-end application results: Given a new query, the model should be able to recommend actions that are of interest to users.

- Diversity: The model should learn a comprehensive set of user intended actions, not just a few common actions.

We examine diversity because it can deepen our understanding of the actions that Web search users most commonly perform, and a diverse set of actions internally could also be

indicative of the ability to perform well on less common queries and on queries whose entities belong to less popular types.

Annotation Guidelines: To measure whether the recommended actions are of interest to users, we adopt a PEGFB graded relevance scale similar to the one used to grade Web search results [30]. In our case, we define the grades as:

- **P**erfect action: Exactly the explicit intent of the user as stated in the query. (only used for queries with context)
- **E**xcellent action: The presumed likely intent of the user as stated in the query.
- **G**ood action: Likely to be interesting to the user, although not the stated intent.
- **F**air action: Possibly of interest to some users who issue the query.
- **B**ad action: Unlikely to be of interest to any user who issues this query.

We employed a total of seven paid independent annotators for grading the actions suggested in each configuration. For each action, two annotations were obtained. Inter-rater agreement using Fleiss' κ was 0.28 (fair agreement) when the P, E, G relevance judgments were collapsed. Note that there is some amount of subjectivity in ratings, especially for queries with no context. For example, on a query for "Obama," some annotators felt that the "Watch videos about" action is *Good*, while others felt it is *Fair*. When exact ratings differed, they still tended to be close in rank. Annotators were also allowed to specify and skip labeling any test query that was judged navigational or that contained entity recognition errors. This occurred in 16.5% of the test cases.

For each query set, each model configuration was set to return up to seven actions to be judged according to our PEGFB scale.

5.5.5 Experimental Results

The results (using P=5, E=4, G=3, F=2, B=1) from our model configurations are summarized in Figure 5.8. The evaluation measure is Normalized Discounted Cumulative Gain

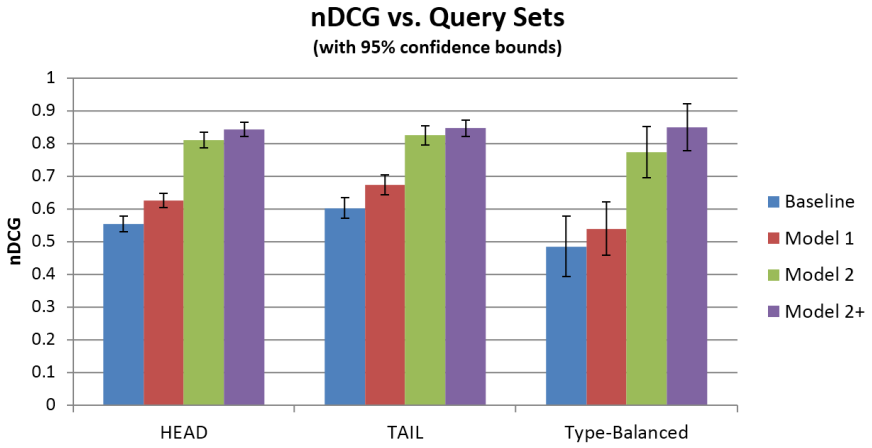

Figure 5.8: **Comparison of Action Models:** Normalized Discounted Cumulative Gain (nDCG) for each experimental configuration from Section 5.5.3, with 95% confidence bounds. The addition of types and entities (Model 2) had the largest effect, followed by clicked hosts (Model 1) and then empty switch (Model 2^+).

(nDCG) on the top-7 suggested actions per model.

For all query sets, addition of the click host (Model 1) improves over the baseline because it provides an additional useful signal for learning accurate clusters. Adding entity type and the entities (Model 2) proves to be the most important signal in terms of significant relevance improvement across evaluation sets. Adding the empty switch (Model 2^+) does not significantly impact overall relevance, however in the Type-Balanced set we see a tendency for this model to perform better. Later in this section, we show that Model 2^+ learns a more diverse set of clusters than other models.

Figure 5.9 shows Mean Relevance as a function of the rank of an action for head queries for Model 2 and Model 2^+. At the top ranks, Model 2^+ is suggesting actions that annotators are generally rating between *Good* and *Excellent*.

Error Analysis

Table 5.2 illustrates action recommendations from our models for the random query "Webster University," which has empty contexts. The Baseline model (which only models action and context) has no information to use for recommending an action other than its ac-

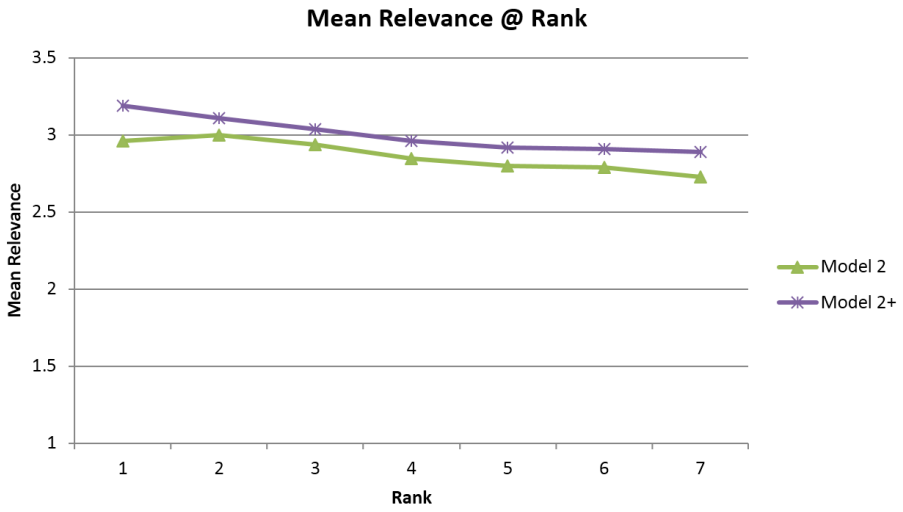

Figure 5.9: **Mean Relevance at Action Rank:** Model 2^+ recommended top actions that had an average annotator rating between *Good* and *Excellent*.

tion priors, and therefore recommends the most popular general actions it learned from its training set.[2] These tend to fit the more common types, so baseline scores are lower on the type-balanced set, which contains fewer common types. Model 1 does a little better by incorporating prior click information, but still recommends actions that do not apply to the entity's type (*e.g.*, "read biography") because the model does not account for type. Models 2 and 2^+ recommend reasonable sets of actions.

One source of error we noticed arose from how the 21 types that we used did not include the primary types of a number of the entities in the data. For example, for the query "Jefferson High School," some of the best actions would be those associated with a *high school* type. However, because *high school* is not among our 21 modeled types, our models recognize "Jefferson High School" only as an *employer* and a *location*. As a result, the recommended actions are more general. It should be possible to alleviate this problem by expanding the number of modeled types.

Note also that among the 21 Freebase entity types we use, some of the types have higher

[2]For the query with context "download Skype," the baseline model is able to recommend actions "download" and "login to."

Baseline (context)	Model 1 (+click)	Model 2 (+type, +entity)	Model 2^+ (+switch)
1. Torrent	1. Torrent	1. Read reviews of	1. Find address
2. Read biography	2. Read biography	2. See map of	2. See pictures of
3. Find adult pics of	3. Read news about	3. Follow sports teams of	3. Find map of
4. Watch videos	4. See pictures of	4. Get weather in	4. Read news about
5. See pictures of	5. Apply for jobs at	5. Apply for jobs at	5. Apply for jobs at
6. Get quotes from	6. Get quotes from	6. Find address of	6. See cost of
7. Apply for jobs at	7. See videos with	7. See tuition of	7. See ranking of

Table 5.2: **Action Recommendation Examples:** Actions recommended by the various models for the query "Webster University". Entity: "Webster University", Context: (∅,∅), Types: *employer*, *university* and *location*.

query log frequency than others. For example, the *person* type has many more entries in the data than the *tourist destination* type. This leads to our models learning action clusters optimized more toward the popular types than the sparser types. We did explore balancing the training data by only keeping elements of the people subtype (*artist*, *politician*, *actor* and *athlete*) with the types fairly equally represented, and found that this led to each of those types having more action diversity. This suggests that to address sparser types, we may want to discover actions based on type-balanced subsets of the data first, and then either use those actions to initialize clusters in the full training, or devise a hierarchical setup that incorporates type-subtype information.

Action Cluster Quality

In addition to the end-to-end application goal, it is also desirable for a model to learn a good, diverse set of actions. One metric for visualizing this is to graph "Total P(Action | Type)" as a function of "Cluster Rank," as in Figure 5.10. This illustrates the distribution of probability mass across the cluster ranks. Here we only compare Models 2 and 2^+, because Model 1 does not model entity type. Given that we used 21 total types, the maximum value would be 2100% (if all 21 types mapped 100% to one cluster). Model 2 appears to have six

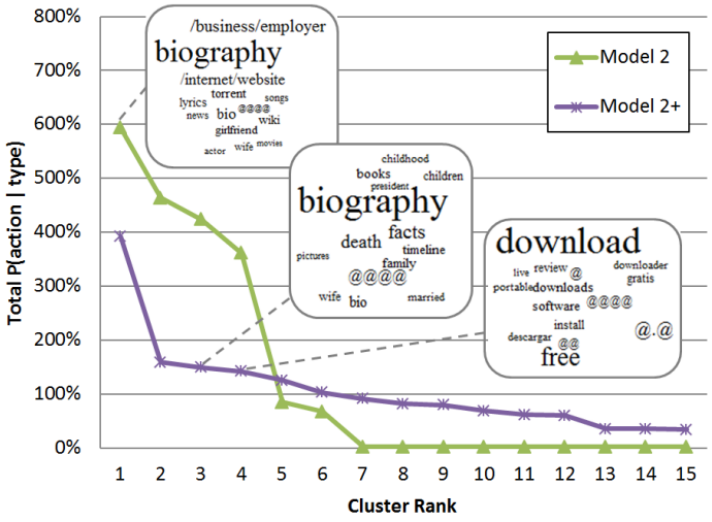

Figure 5.10: **Action Cluster Quality:** Model 2^+ distributes probability of action given type more evenly across actions than Model 2.

primary action clusters that receive the majority of the probability from types, while Model 2^+ learns a much more diverse set of actions clusters, which we also observed by inspecting the word clouds in the Φ parameter.

Note that only learning 6 primary action clusters does not mean that Model 2 can only recommend up to 6 distinct action phrases. First, the remaining clusters do have nonzero weight and can contribute action phrases. Second, individual action clusters may contain a mixture of action phrases. For example, one of the Model 2 clusters contains actions for "read biography," "find lyrics," and "download file" all within the same cluster. This does not cause type mismatches at decoding time because action phrases are typed (*e.g.*, "download file" will only be recommended when the entity is of a type it applies to, such as *software* type), but it does limit the ability of the models to discover and refine good action clusters specifically around the less common actions. The lower ranked clusters within Model 2^+ do look very coherent around specific actions, for example, "read biography" is in a cluster only with related terms such as "facts," "childhood" and "timeline," while "download" is in a cluster with related terms like "software," "install" and "free."

5.6 Conclusions

In this chapter we proposed the notion of *Actions for Web search*. We conducted an annotation study on query log data to gauge the prevalence of entities and associated actions in search. We developed generative models to learn latent actions from queries, and we implemented them over large real-world query logs. We experimentally showed that modeling click hosts and entity types, along with query context words, yields high relevance on the task of action recommendation, and that explicitly representing empty contexts greatly improves action diversity. Finally, we addressed various issues for developing an end-to-end system for actions, and we are now able to automatically recommend good sets of actions for users issuing new queries.

Future directions for this line of work include expanding the number of entity types and modeling actions for "entity category" queries (*e.g.*, "shoes"). Additionally, we believe that our current random initialization of action clusters can be improved upon by seeding the clusters with some small amount of prior knowledge. We are also considering adding a user model to our approach in order to better target user-specific actions. For the "Webster University" query in Table 5.2, for example, actions such as "read reviews of" and "see ranking of" are more suited for prospective students, while "see map of" and "follow sports teams of" are a better fit for current students.

This chapter takes first steps towards a larger vision of search as an action broker. We envision a world where publishers can tag (automatically or manually) their Web pages and native applications with the actions that they can accomplish; a world where users' intended actions can be inferred and executed seamlessly via connections to these providers.

The overall hypothesis of this thesis is that domain-independent Web text processing techniques can leverage large knowledge bases such as Freebase and Wikipedia to better understand and apply the diverse information expressed in Web text. In previous chapters we demonstrated this multiple times in the context of domain-independent information extraction, and in this chapter we further applied our techniques to a Web search scenario. Whereas domain-independent Web text processing without knowledge bases paved the way for generally capturing and interacting with information in Web text, our research promises

that the careful addition of large knowledge bases can enable a much richer future, featuring a deeper understanding of information on the Web, stronger performance on end-tasks applying this information, and exciting new experiences enabled for both AI and human users.

Chapter 6

FUTURE WORK

Four areas of future work that would be valuable to explore include:

The Full Universe of Structured Data: It would be difficult for domain-specific KBs to benefit open-domain Web text processing in any general way by themselves, but what if they were used in large numbers to augment (rather than replace) the large, general KBs that we use? One very compelling direction of future work is to see how our work scales when many additional KBs are integrated into our large, general KBs. An immediate example would be all the other Wikis that people have created online for their favorite television shows, games, and other interests. While Wikipedia contains only 4 million entities, the Wikia family of sites contains an additional 20 million entities that could be incorporated.[1] Incorporating sources such as Wikia, Linked Open Data [9] or commercial knowledge bases [112] would present many exciting challenges and opportunities.

Stronger Handling of Ambiguous Strings: While Chapter 4 presented initial methods for disambiguating surface strings in a number of cases, there do remain some extremely challenging cases that we cannot fully handle at present time. One prime example is common people names. A name such as "John Smith" refers to over a hundred people in Wikipedia and also many thousands of people that are not prominent enough to be in Wikipedia. When we see text like "John Smith scored 92 on the exam" on the Web, in many cases there would not even be enough context for human experts to correctly pick which "John Smith." There are numerous ways to start approaching this problem, such as borrowing ideas from cross-document coreference [115], exploring how resources the style and scale of Facebook and LinkedIn can be integrated, or even commonsense reasoning using background knowledge.

Dynamic Integration of Types: Throughout this thesis we use the Freebase type

[1]The *Wikia sites* include thousands of Wikis hosted at www.wikia.com

system which contains 1,000 to 2,000 types. Freebase types offer good coverage over all the entities within Wikipedia and Freebase and generally served us well for our tasks. However, we did encounter scenarios where it would have helped to have more types. For example, in Chapter 5 when we see an entity like "Jefferson High School", our techniques would be able to provide the best experience if we could identify it as a *high school*. While Freebase has types for *school* and *organization*, it does not recognize *high school* as a separate type. Using all the Wikipedia categories instead is not the solution because most Wikipedia categories (*e.g.*, "Educational institutions established in 1985") are too fine-grained to be useful here. What would help is if we could dynamically integrate useful types into the system. These additional useful types might be identified from query logs, hypernym patterns [43], or by methods to identify the most useful subsets of Wikipedia categories. An ability to dynamically integrate types would also help to maintain overall system domain-independence when new types arise in the future.

Further Application: Open-domain Web text processing, especially with the advances leveraging KBs described in this thesis, is at a stage where it can provide practical benefit for applications. In this work we demonstrated multiple applications such as typed question answering and actions for Web search. Our last item for future directions is to push these benefits of Web text processing with KBs to increasingly more applications and system components, with the goal of empowering better user applications and smarter artificial intelligence programs for the future.

Chapter 7

CONCLUSIONS

The Web is the largest collection of text in human history, and a growing line of research is to extract and process as much of its information as possible by using relation and domain independent techniques such as Open IE. To handle the full diversity of knowledge on the Web, these techniques have typically avoided fixed knowledge bases that would limit the vocabulary that they can process. In this thesis, we presented that certain KBs like Freebase and Wikipedia now have enough coverage of general knowledge that they can be effectively leveraged in open-domain Web text processing, both to improve task results and enable new functionality. We demonstrated this over a number of tasks including identifying interesting assertions, identifying functional relations, open entity linking, and actions for Web search.

Contributions of this work include:

Interesting Assertions: After noticing that the TextRunner Open IE system extracts many uninteresting extractions, we formulated the new problem of automatically classifying extraction interestingness. We developed several practical models of interestingness that, when implemented as filters, offer substantial improvements over the prior technique of sorting assertions by frequency. Our most effective model leveraged information from Wikipedia Infoboxes, demonstrating a way to use KBs in this task. We reported on the first study of interestingness in extraction. Among other findings, we show that our filtering significantly improves the fraction of interesting results contained within TextRunner's top thirty results from 41.6% interesting to 64.1% interesting.

Relation Functionality: To study what deeper understanding can be gained about relation phrases appearing in Web text, we analyzed the problem of relation functionality. We identify several linguistic phenomena that make the problem of corpus-based functionality identification surprisingly difficult. We designed and implemented three novel techniques for identifying functionality based on instance-based counting, distributional differences, and

use of external knowledge bases. Our best method outperforms the existing approaches by wide margins, increasing area under the precision-recall curve from 0.61 to 0.88.

Entity Linking: We proposed novel techniques around collective contexts and inlink ratio that allow a large corpus of Web extractions to be entity linked both quickly and accurately. Using these techniques, we linked entities from millions of extractions into Wikipedia. We then motivated and introduced the *unlinkable noun phrase problem*. We proposed a novel method for discriminating entities from arbitrary noun phrases, utilizing features derived from Google Books ngrams, and also adapted and scaled instance-to-instance class propagation in order to associate types with non-Wikipedia entities. We implemented and evaluated our proposed methods for unlinkable noun phrases, empirically verifying significant improvement over appropriate baselines. Last, we offered initial solutions for how to handle the challenge of ambiguous surface strings.

Actions for Web Search: To demonstrate how KBs can offer benefit to Web text processing even beyond extraction, we introduced the new paradigm of *Actions for Web search*. We conducted an annotation study to established that there are specific Web actions that users aim to perform on entities. We proposed probabilistic models to generate entity bearing queries from actions, incorporating information from signals such as context words, clicked hosts, and entity types. We trained our models on three months of query data from a commercial search engine, and address the necessary end-to-end system issues for producing a system to recommend suitable actions for new queries. We conducted a user-study to evaluate our different models, showing which model components are most important for generating actions.

Overall: Through our series of studies, we observed numerous times that while open-domain Web text processing can extract vast amounts of general information from noisy Web text, these processes can benefit greatly by leveraging knowledge bases like Freebase and Wikipedia. This applies even to Web text involving relationships and entities of which the KBs had no prior knowledge. Our work advances the levels of performance and understanding that are possible with open-domain Web text processing, and brings us a few steps closer to the compelling vision promised by machine reading of Web text.

BIBLIOGRAPHY

[1] Eneko Agirre and David Martinez. Integrating selectional preferences in wordnet. In *Proceedings of the first International WordNet Conference*, 2002.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of ISWC*, 2007.

[3] N. Balasubramanian and S. Cucerzan. Topic pages: An alternative to the ten blue links. In *IEEE-ICSC*, 2010.

[4] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the Web. In *Proceedings of IJCAI*, 2007.

[5] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2008.

[6] Michele Banko. *Open Information Extraction for the Web*. PhD thesis, University of Washington, 2009.

[7] Robert Bart. Design document for scalable unlinkable entities type-prediction system. In *University of Washington Internal Research Report*, 2012.

[8] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. HarperInformation, 2000.

[9] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. In *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[10] D.M. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. In *Journal of Machine Learning Research*, 2003.

[11] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD '08*, pages 1247–1250, New York, NY, USA, 2008.

[12] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117. Elsevier Science Publishers B. V., 1998.

[13] A. Broder. A taxonomy of web search. In *SIGIR Forum*, 2002.

[14] A. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of SIGIR*, 2007.

[15] Razvan Bunescu and Marius Paşca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the 11th Conference of the European Chapter of the Association of Computational Linguistics (EACL)*, 2006.

[16] Davide Buscaldi and Paolo Rosso. Mining knowledge from wikipedia for the question answering task. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, 2006.

[17] Andrew Carlson. *Coupled Semi-Supervised Learning.* PhD thesis, Carnegie Mellon University, 2010.

[18] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of AAAI*, 2010.

[19] M.J. Carman, F. Crestani, M. Harvey, and M. Baillie. Towards query log based personalization using topic models. In *Proceedings of CIKM*, 2010.

[20] G. J. Chaitin. Register allocation & spilling via graph coloring. In *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, SIGPLAN '82, pages 98–105, New York, NY, USA, 1982. ACM.

[21] Janara Christensen and Marius Pasca. Instance-driven attachment of semantic annotations over conceptual hierarchies. In *Proceedings of EACL*, 2012.

[22] S. Colton and A. Bundy. On the notion of interestingness in automated mathematical discovery. In *Proceedings of AISB*, 1999.

[23] Silviu Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of EMNLP*, 2007.

[24] A. Culotta, A. McCallum, and J. Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *HLT-NAACL*, 2006.

[25] J. R. Curran and S. Clark. Language independent NER using a maximum entropy tagger. In *Proceedings of CoNLL*, 2003.

[26] N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu. A web of concepts. In *Proceedings of PODS*, 2009.

[27] D. Downey, O. Etzioni, and S. Soderland. A probabilistic model of redundancy in information extraction. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[28] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. Entity disambiguation for knowledge base population. In *Proceedings of COLING*, 2010.

[29] T. Dunning. Accurate methods for the statistics of surprise and coincidence. In *Computational Linguistics*, volume 19, 1993.

[30] G. Dupret and B. Piwowarski. A User Behavior Model for Average Precision and its Generalization to Graded Judgments. In *Proceedings of SIGIR*, 2010.

[31] P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. In *Acta Mathematica Academiae Scientiarum Hungaricae*, 1966.

[32] 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*, 165(1):91–134, 2005.

[33] Oren Etzioni, Michele Banko, and Michael J. Cafarella. Machine Reading. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[34] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam. Open information extraction: The second generation. In Toby Walsh, editor, *IJCAI*, pages 3–10. IJCAI/AAAI, 2011.

[35] James A. Evans and Jacob G. Foster. Metaknowledge. In *Science*, 2011.

[36] A. Fader, S. Soderland, and O. Etzioni. Scaling wikipedia-based named entity disambiguation to arbitrary web text. In *WikiAI 2009*, 2009.

[37] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of EMNLP*, 2011.

[38] Paolo Ferragina and Ugo Scaiella. Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of CIKM*, 2010.

[39] J. Gao, K. Toutanova, and W. Yih. Clickthrough-based latent semantic models for web search. In *Proceedings of SIGIR*, 2011.

[40] N. Guarino and C. Welty. An overview of OntoClean. In *Handbook of Ontologies in Information Systems*, pages 151–172, 2004.

[41] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proceedings of SIGIR*, 2009.

[42] G. Hamerly and C. Elkan. Learning the k in k-means. In *Proceedings of the 7th Annual Conference on Neural Information Processing Systems (NIPS)*, 2003.

[43] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING*, 1992.

[44] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Proceedings of EMNLP*, 2011.

[45] R. Hoffmann, S. Amershi, K. Patel, F. Wu, J. Fogarty, and D. Weld. Amplifying community content creation using mixed-initiative information extraction. In *Proceedings of CHI*, 2009.

[46] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. Learning 5000 relational extractors. In *Proceedings of ACL*, 2010.

[47] J. Hopcraft and R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378, 1973.

[48] Eduard Hovy, Zornitsa Kozareva, and Ellen Riloff. Toward completeness in concept extraction and classification. In *Proceedings of EMNLP*, 2009.

[49] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. In *The Computer Journal*, 1999.

[50] A. Jain and M. Pennacchiotti. Domain-independent entity extraction from web search query log. In *Proceedings of WWW*, 2011.

[51] B.J. Jansen, D. Booth, and A. Spink. Determining the user intent of web search engine queries. In *Proceedings of WWW*, 2007.

[52] T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley and Sons, 1995.

[53] G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *Proceedings of ICDE*, 2008.

[54] M. Keller, S. Bengio, and S.Y. Wong. Benchmarking non-parametric statistical tests. In *Advances in Neural Information Processing Systems (NIPS) 18*, 2005.

[55] C. Kemke and E. Walker. Planning with action abstraction and plan decomposition hierarchies. In *Proceedings of IAT*, 2006.

[56] J. Kim and D. Moldovan. Acquisition of semantic patterns for information extraction from corpora. In *Procs. of Ninth IEEE Conference on Artificial Intelligence for Applications*, pages 171–176, 1993.

[57] K. Kipper-Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. In *Ph.D. thesis. University of Pennsylvania*, 2005.

[58] Z. Kozareva and E. Hovy. Learning arguments and supertypes of semantic relations using recursive patterns. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2010.

[59] Zornitsa Kozareva, Konstantin Voevodski, and Shang-Hua Teng. Class label enhancement via related instances. In *Proceedings of EMNLP*, 2011.

[60] Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. Collective annotation of wikipedia entities in text. In *Proceedings of KDD*, 2009.

[61] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[62] Mirella Lapata. The web as a baseline: Evaluating the performance of unsupervised web-based models for a range of nlp tasks. In *In Proceedings of HLT*, pages 121–128, 2004.

[63] Changki Lee, Yi-Gyu Hwang, and Myung-Gil Jang. Fine-grained named entity recognition and relation extraction for question answering. In *Proceedings of SIGIR*, 2007.

[64] Douglas B. Lenat. Cyc: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, November 1995.

[65] T. Lin, O. Etzioni, and J. Fogarty. Identifying interesting assertions from the web. In *Proceedings of CIKM*, 2009.

[66] T. Lin, Mausam, and O. Etzioni. Commonsense from the web: Relation properties. In *AAAI Fall Symposium on Commonsense*, 2010.

[67] Thomas Lin, Oren Etzioni, and James Fogarty. Filtering information extraction via user-contributed knowledge. In *Proceedings of WikiAI*, 2009.

[68] Thomas Lin, Mausam, and Oren Etzioni. Identifying functional relations in web text. In *Proceedings of EMNLP*, 2010.

[69] Thomas Lin, Mausam, and Oren Etzioni. Entity linking at web scale. In *Knowledge Extraction Workshop (AKBC-WEKEX)*, 2012.

[70] Thomas Lin, Mausam, and Oren Etzioni. No noun phrase left behind: Detecting and typing unlinkable entities. In *Proceedings of EMNLP*, 2012.

[71] Thomas Lin, Patrick Pantel, Michael Gamon, Anitha Kannan, and Ariel Fuxman. Active objects: Actions for entity-centric search. In *Proceedings of WWW*, 2012.

[72] Xiao Ling and Daniel S. Weld. Fine-grained entity recognition. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*, 2012.

[73] B. Liu, W. Hsu, S. Chen, and Y. Ma. Analyzing the subjective interestingness of association rules. In *IEEE Intelligent Systems 15*, 2000.

[74] Metaweb Technologies. Freebase data dumps. In *http://download.freebase.com/datadumps/*, 2009.

[75] T. Metzinger and V. Gallese. The emergence of a shared action ontology: Building blocks for a theory. In *Consciousness and Cognition, 12*, 2003.

[76] Jean-Baptiste Michel, Yuan Kui Shen, Aviva P. Aiden, Adrian Veres, Matthew K. Gray, The Google Books Team, Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, and Steven Pinker. Quantitative analysis of culture using millions of digitized books. In *Science*, 2010.

[77] Rada Mihalcea and Andras Csomai. Wikify!: linking documents to encyclopedic knowledge. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 233–242, New York, NY, USA, 2007. ACM.

[78] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[79] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17h ACM International Conference on Information and Knowledge Management (CIKM)*, 2008.

[80] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 227–236, New York, NY, USA, 2011. ACM.

[81] Ndapandula Nakashole and Gerhard Weikum. Real-time population of knowledge bases: Opportunities and challenges. In *Knowledge Extraction Workshop (AKBC-WEKEX)*, 2012.

[82] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *Proceedings of EMNLP*, 2012.

[83] Ryan Oman. Fast identification of candidate entities for context disambiguation. In *University of Washington Undergraduate Research Report*, 2011.

[84] Marius Paşca. Acquisition of categorized named entities for web search. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2004.

[85] P. Pantel and A. Fuxman. Jigs and lures: Associating web queries with structured entities. In *Proceedings of ACL*, 2011.

[86] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of EMNLP*, 2009.

[87] Patrick Pantel, Thomas Lin, and Michael Gamon. Mining entity types from query logs via user intent modeling. In *Proceedings of ACL*, 2012.

[88] Patrick Pantel and Deepak Ravichandran. Automatically labeling semantic classes. In *Proceedings of HLT-NAACL*, 2004.

[89] Danuta Ploch. Exploring entity relations for named entity disambiguation. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2011.

[90] H. Poon, J. Christensen, P. Domingos, O. Etzioni, R. Hoffmann, C. Kiddon, T. Lin, X. Ling, Mausam, A. Ritter, S. Schoenmackers, S. Soderland, D. Weld, F. Wu, and C. Zhang. Machine reading at the university of washington. In *Proceedings of FAM-LbR*, 2010.

[91] A-M. Popescu. Information extraction from unstructured web text. In *Ph.D. thesis. University of Washington*, 2007.

[92] J. Prager, S. Luger, and J. Chu-Carroll. Type nanotheories: a framework for term comparison. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM)*, 2007.

[93] C. Price and K. Spackman. SNOMED clincal terms. In *British Journal of Healthcare Computing & Information Management*, volume 17, 2000.

[94] Willard Van Orman Quine. On what there is. In *Review of Metaphysics*, 1948.

[95] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[96] Altaf Rahman and Vincent Ng. Inducing fine-grained semantic classes via hierarchical and collective classification. In *Proceedings of COLING*, pages 931–939, 2010.

[97] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*, 2009.

[98] Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2011.

[99] E. Riloff. Automatically constructing extraction patterns from untagged text. In *Procs. of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1044–1049, 1996.

[100] A. Ritter, D. Downey, S. Soderland, and O. Etzioni. It's a contradiction - no, it's not: A case study using functional relations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[101] A. Ritter, Mausam, and O. Etzioni. A latent dirichlet allocation method for selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2010.

[102] D. E. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of WWW*, 2004.

[103] M. Sadoski, E.T. Goetz, and J.B. Fritz. A causal model of sentence recall: Effects of familiarity, concreteness, comprehensibility and interestingness. In *Journal of Reading Behavior, 25*, 1993.

[104] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. In *Information Processing and Management*, pages 513–523, 1988.

[105] C. Sauper and R. Barzilay. Automatically generating wikipedia articles: A structure-aware approach. In *Proceedings of ACL*, 2009.

[106] S. Schoenmackers, O. Etzioni, and D. Weld. Scaling textual inference to the web. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[107] Stefan Schoenmackers, Oren Etzioni, Daniel S. Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *Proceedings of EMNLP*, 2010.

[108] G. Schraw. Situational interest in literary text. In *Contemporary Educational Psychology, 22*, 1997.

[109] S. Sekine and H. Suzuki. Acquiring ontological knowledge from query logs. In *Proceedings of WWW*, 2007.

[110] Satoshi Sekine and Chikashi Nobata. Definition, dictionaries and tagger for extended named entity hierarchy. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*, 2004.

[111] Yusuke Shinyama and Satoshi Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 304–311, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[112] Avirup Sil, Ernest Cronin, Penghai Nie, Yinfei Yang, Ana-Maria Popescu, and Alexander Yates. Linking Named Entities to Any Database. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2012.

[113] P.J. Silvia. *Exploring the Psychology of Interest.* Oxford University Press, New York, NY, 2006.

[114] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. In *Proceedings of ODBASE*, 2002.

[115] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of ACL*, 2011.

[116] Valentin I. Spitkovsky and Angel X. Chang. A cross-lingual dictionary for english wikipedia concepts. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[117] P. Srinivasan and A. Yates. Quantifier scope disambiguation using extracted pragmatic knowledge: Preliminary results. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2009.

[118] Yohei Takaku, Nobuhiro Kaji, Naoki Yoshinaga, and Masashi Toyoda. Identifying constant and unique relations by using time-series text. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 883–892, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[119] Partha Pratim Talukdar and Fernando Pereira. Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2010.

[120] Partha Pratim Talukdar, Joseph Reisinger, Marius Paşca, Deepak Ravichandran, Rahul Bhagat, and Fernando Pereira. Weakly supervised acquisition of labeled class instances using graph random walks. In *Proceedings of EMNLP*, 2008.

[121] Partha Pratim Talukdar, Derry Tanti Wijaya, and Tom Mitchell. Coupled temporal scoping of relational facts. In *Proceedings of WSDM*, 2012.

[122] J. Volker, D. Vrandecic, and Y. Sure. Automatic evaluation of ontologies (AEON). In *Proceedings of the 4th International Semantic Web Conference (ISWC)*, 2005.

[123] Chi Wang, Kaushik Chakrabarti, Tao Cheng, and Surajit Chaudhuri. Targeted disambiguation of ad-hoc, homogeneous sets of named entities. In *Proceedings of the 21st International World Wide Web Conference (WWW)*, 2012.

[124] Daniel S. Weld, Raphael Hoffmann, and Fei Wu. Using wikipedia to bootstrap open information extraction. In *SIGMOD Record*, 2008.

[125] Derry Tanti Wijaya and Reyyan Yeniterzi. Understanding semantic changes of words over centuries. In *Workshop on Detecting and Exploiting Cultural Diversity on the Social Web*, 2011.

[126] Wikipedia. Wikipedia: The free encyclopedia. In *http://www.wikipedia.org*. Wikimedia Foundation, 2004.

[127] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[128] F. Wu and D. Weld. Autonomously semantifying wikipedia. In *Proceedings of CIKM*, 2007.

[129] H. Yao and H. Hamilton. Mining functional dependencies from data. In *Data Mining and Knowledge Discovery*, 2008.

[130] A. Yates and O. Etzioni. Unsupervised methods for determining object and relation synonyms on the web. In *Journal of Artificial Intelligence Research*, volume 34, pages 255–296, 2009.

[131] X. Yin and S. Shah. Building taxonomy of web search intents for name entity queries. In *Proceedings of WWW*, 2010.

[132] H. Zaragoza, H. Rode, P. Mika, J. Atserias, M. Ciaramita, and G. Attardi. Ranking very many typed entities on wikipedia. In *Proceedings of CIKM*, 2007.

[133] Yiping Zhou, Lan Nie, Omid Rouhani-Kalleh, Flavian Vasile, and Scott Gaffney. Resolving surface forms to wikipedia topics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1335–1343, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Appendix A

DEMONSTRATION OF INTERESTINGNESS FILTER

As an example, Table A.1 shows the top 30 TextRunner Web extractions (by frequency) before and after applying our interestingness filter. We see that the original set of Web extractions contains many uninteresting or uninformative extractions such as:

- "Brazil is the only country"
- "Brazil can find basic info"
- "Brazil is the team"
- "Brazil is the heavy favorite"
- "Brazil has more time"
- "Brazil is one country"
- "BRAZIL comes to POCONOS.Post"

By filtering out the uninteresting extractions, many of the more interesting extractions are able to come to the top. After filtering, we observe numerous extractions that will be more interesting to people, such as:

- "Brazil is the size of the United States"
- "Brazil is the best soccer team"
- "Brazil has the world s largest Catholic population"
- "Brazil has won the World Cup four times"
- "Brazil produces approximately 4 billion gallons of ethanol"
- "Brazil is the largest producer of sugarcane"

"Brazil" Web extractions (unfiltered)	"Brazil" Web extractions (with *interestingness* filter)
1. Brazil is the only country	1. Brazil creates buffer zone
2. Brazil can find basic info	2. Brazil is the size of the United States
3. Brazil is the team	3. Brazil is the best soccer team
4. Brazil is not the only country	4. Brazil is the largest investor
5. Brazil has the largest economy	5. Brazil has the world 's largest Catholic population
6. Brazil has the best players	6. Brazil has won the World Cup four times
7. Brazil played a key role	7. Brazil produces approximately 4 billion gallons of ethanol
8. Brazil is the heavy favorite	8. Brazil offers great profit opportunities
9. Brazil has great potential	9. Brazil to open ethanol embassy
10. Brazil has a problem	10. Brazil is the second largest producer of soy
11. Brazil is the largest producer	11. Brazil seeks Lat Am 's first nuke sub
12. Brazil lost to France	12. Brazil to make 2014 World Cup bid
13. Brazil won the match	13. Brazil hosted the 1950 World Cup
14. Brazil is the world	14. Brazil is the largest producer of sugarcane
15. Brazil has won the World Cup	15. Brazil won $ 200 million order
16. Brazil has laws	16. Brazil won the Beach Soccer World Cup
17. Brazil to influence US design	17. Brazil creates new protected areas
18. Brazil has made great strides	18. Brazil is the world 's top sugar producer and exporter
19. Brazil is a Portuguese colony	19. Brazil has the largest Black population
20. Brazil has a comparative advantage	20. Brazil has the largest population of Japanese outside of Japan
21. Brazil to win the World Cup	21. Brazil has a very complex society
22. Brazil beat Italy	22. Brazil has won the World Cup more times
23. Brazil has more time	23. Brazil produces the same amount of ethanol
24. Brazil will win this World Cup	24. Brazil needs outside aviation help
25. Brazil speaks Portuguese	25. Brazil has discovered huge new petroleum reserves
26. Brazil is one country	26. Brazil recognized China 's market economy status
27. Brazil has more chances	27. Brazil attracted over 5 million European and Asian immigrants
28. Brazil has played an important role	28. Brazil to mull forex steps
29. BRAZIL comes to the POCONOS.Post	29. Brazil is the world 's largest beef producer
30. Brazil is time	30. Brazil is the second largest exporter of soybeans

Table A.1: **Interestingness Filter:** The top 30 Web extractions for "Brazil" after filtering *(right)* are more *interesting* than the top 30 Web extractions before filtering *(left)*.

Appendix B

FUNCTIONAL RELATION PHRASES

Table B.1 shows 15 examples of relation phrases from a Web text corpus that our system determined to be functional. If a relation is functional then it maps each first argument to at most one second argument. For example, if Brazil "is the largest country in" South America, then it cannot also be the largest country in any other continent.

Relation Phrase	Example Instances from Web Text
(*, is a trademark of, <company>)	(Teflon → DuPont)
(*, is the largest country in, <continent>)	(Brazil → South America)
(*, is the highest mountain in, <continent>)	(Mt Kilimanjaro → Africa)
(*, is the capital city of, <country>)	(Amsterdam → Netherlands)
(*, is the national airline of, <country>)	(Aer Lingus → Ireland)
(*, is the brand name for, <drug>)	(Elavil → Amitriptyline)
(*, is a registered trademark of, <film distributor>)	(Shrek → Dreamworks)
(*, was a powerful, <gender>)	(Medea → female)
(*, is the birthstone for, <month>)	(Alexandrite → June)
(*, is the largest moon of, <planet>)	(Ganymede → Jupiter)
(*, is an extension of, <programming language>)	(AdabasTcl → Tcl)
(*, is the governing body of, <sport>)	(Motocyclisme → motorcycle racing)
(*, is the second largest lake in, <state>)	(Lake Livingston → Texas)
(*, is the sequel to, <videogame>)	(Raystorm → Rayforce)
(*, is an organization established in, <year>)	(Unity Enterprise → 1989)

Table B.1: **Functions Examples:** Examples of relation phrases that our system identified to be *functional*.

Appendix C

MOST COMMON LINKED ENTITIES

To provide a better sense of the entities and extractions we work with, Table C.1 shows the Wikipedia/Freebase entities that were linked to from the greatest number of Web extractions. This data is from our entity linking of 15 million high quality extractions from the ReVerb Open Information Extraction system.

Entity	Freebase ID	Extractions	Example Web Extraction
Barack Obama	02mjmr	16,094	(Obama, was born in, Hawaii)
Jesus	045m1_	12,051	(Jesus, was born in, Bethlehem)
Internet	03rlt	8,194	(The Internet, offers a wealth of, information)
United States	09c7w0	8,002	(the United States, is a nation of, immigrants)
China	0d05w3	7,790	(China, has a population of, 1.3 billion)
Washington DC	0rh6k	7,727	(D.C., is not, a state)
Apple Inc	0k8z	7,686	(Apple, has sold, 100 million iPods)
George W Bush	09b6zr	7,682	(Bush, was born in, New Haven)
Company	03bxgrp	7,039	(Companies, are not, charities)
Water	0838f	6,993	(Water, is essential for, life)
Google	045c7b	6,629	(Google, bought YouTube for, $1.65 billion)
Israel	03spz	6,561	(Israel, is the size of, New Jersey)
Music	04rlf	6,345	(Music, is the rhythm of, Life)
India	03rk0	6,237	(India, is predominantly, Hindu)
Bill Clinton	0157m	5,710	(Clinton, was elected governor of, Arkansas)

Table C.1: **Top Linked Entities:** Prominent knowledge base entities such as Barack Obama are the most common among our linked extractions.

Appendix D

EXAMPLES OF IDENTIFIED UNLINKABLE ENTITIES

A central emphasis of our work is the ability to work with *all* entities on the Web, not just the prominent entities that can be found in KBs such as Wikipedia and Freebase. Chapter 4 presents a method for predicting *semantic types* of general non-KB entities by leveraging the KB entities. Table D.1 below shows type predictions that our system made for some example non-Wikipedia noun phrases that it found in Web extraction data.

Predicted Type	Unlinkable Noun Phrases from the Web
Band	"Ctrl-Alt-Del," "White Rhino," "Boxhead,"
	"Descarga," "Lightswitch," "Madmartigan"
Food	"prune juice," "wheatgrass juice," "rabbit meat,"
	"goat milk," "dried plums," "pastry cream"
Company	"Bamboo Solutions," "Telephia," "Zurvita,"
	"Advanstar," "FatCow," "SFI Electronics"
Computer/Video Game	"Superstar Chefs," "Slingo Supreme," "Docker Sokoban,"
	"Colin McRae Rally 2005," "Neverland Card Battles"
Quotation Subject	"hard times," "good attitude," "humbleness,"
	"inaction," "true beauty," "having kids"
Person	"David Enders," "Elizabeth Martin," "Alex Carr,"
	"Aileen Gallagher," "Ellen Kanner," "Ted Poulos"
etc.	"asbestos exposure" (medical risk), "fashion designer"
	(job title), "Bradley Hospital" (hospital)

Table D.1: **Example Unlinkable Entities:** Examples of unlinkable (non-Wikipedia) entities that our system detected in Web extraction data and correctly typed.

Appendix E

EXAMPLES OF ACTION CLUSTERS

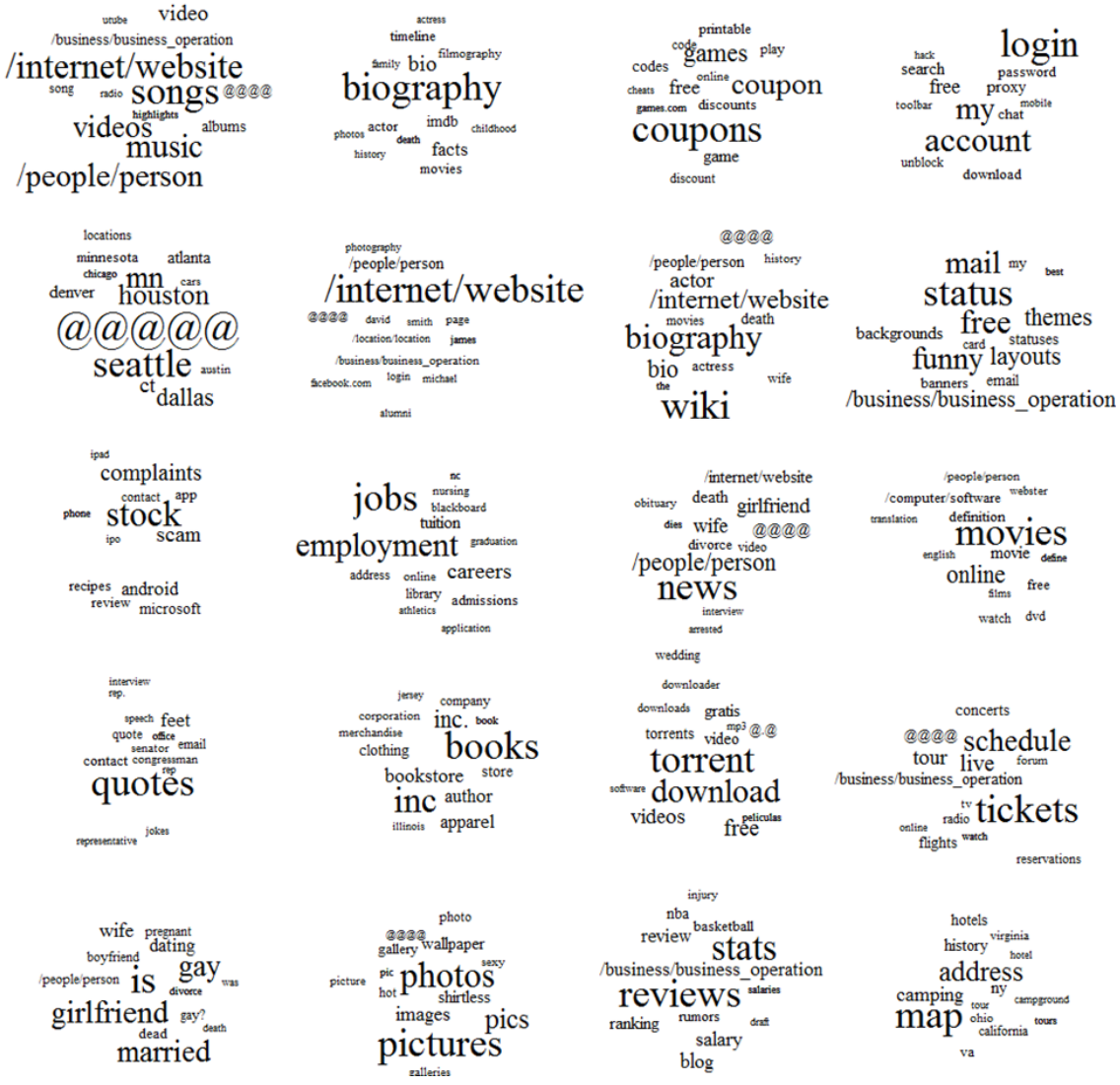

Figure E.1: **Action Clusters:** Key user actions (*e.g.*, "read biography," "find coupons") can be identified from the P(query context words | latent action cluster) parameter.

VITA

Thomas Lin received a B.S. degree in Computer Science and Engineering and an M.Eng. degree in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 2003 and 2004, respectively. He continued his studies at the University of Washington, where he was advised by Professor Oren Etzioni and Professor Mausam. While conducting his studies he was supported by a National Defense Science and Engineering Graduate fellowship. He received an M.S. degree in 2009, and a Ph.D. degree in 2012, both in Computer Science from the University of Washington.