Machine Reading: from Wikipedia to the Web

Fei Wu

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

Doctor of Philosophy

University of Washington

2010

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Fei Wu

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Daniel S. Weld

Reading Committee:

_____

Daniel S. Weld

_____

Oren Etzioni

_____

Pedro Domingos

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Machine Reading: from Wikipedia to the Web

Fei Wu

Chair of the Supervisory Committee:
Professor Daniel S. Weld
Computer Science and Engineering

Berners-Lee's compelling vision of a Semantic Web is hindered by a chicken-egg problem, which can be best solved via machine reading — automatically extracting information from natural-language texts to make them accessible to software agents. We argue bootstrapping is the best way to build such a system. We choose Wikipedia as an initial data source, because it is comprehensive, high-quality, and contains enough collaboratively-created structure to launch a self-supervised bootstrapping process. We have developed three systems that realize our vision:

- KYLIN, which applies Wikipedia heuristic of matching sentences with infoboxes to create training examples for learning relation-specific extractors.

- KOG, which automatically generates Wikipedia Infobox Ontology by integrating evidence from heterogeneous resources via joint inference using Markov Logic Networks.

- WOE, which uses Wikipedia heuristic to create matching sentence set as done in KYLIN, but it abstracts these examples to relation-independent training data to learn an unlexicalized open extractor.

The results of our experiments show that these automatically learned systems can render much of Wikipedia into high-quality semantic data, which provides a solid base to bootstrap toward the general Web.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGMENTS

This work was impossible without the help of many people. I would like to begin by thanking my student collaborators. Raphael Hoffmann and Hoifung Poon are extremely smart and a great pleasure to work with. Raphael built a great data platform which benefits many students including me; Chapter 4 of this dissertation includes a number of his contributions. My first research project at UW was done with Hoifung; ever since then, I have benefited from our meeting and discussions from time to time. I would like to thank everyone in the KnowItAll group at UW — especially Michele Banko, Alan Ritter, Matthew Broadhead, Mausam, Stephen Soderland, Stef Schoenmackers, Michael Cafarella and Alex Yates. Michele, Alan and Matthew deserve special thanks for helping on the TextRunner code and proving valuable insights on my projects.

I am grateful to the UW Computer Science and Engineering Department. All faculty members, staff and students made my 5-year study in this department a wonderful journey. In particular, Lindsay Michimoto helped me maintain a smooth and enjoyable graduate school life by always quickly clearing my concerns and answering my questions. Henry Kautz and Pedro Domingos gave me invaluable advice on how to identify and tackle great problems when advising me on my first research project. Oren's incisive questions deepened my understanding on my work. His suggestions for future directions enlightened my thoughts, and his advice on presentation strategies were always helpful.

I was lucky to intern at IBM Almaden Research Center and Google. These experience gave me great opportunity to apply what I learned at graduate school to solve practical problems, and also let me experience industry lives. I would like to thank my mentors John Tang, Tessa Lau at IBM, and Alon Halevy, Jayant Madhavan at Google. They helped to set up interesting research projects, and guided me through the whole process of clarifying key challenges, proposing solutions, drilling down into the finest experiment details, and

writing technical papers.

I would like to thank my advisor at UW, Dan Weld. He offered me mentorship toward becoming a great researcher and a great person. He made it top priority to help his students to reach their full potential, and was always there to offer the guidance when it's needed. He taught me how to distill important and challenging problems to work on, and brought together all the resources to help me tackle them. He made me feel to be in charge of my research projects, while continuously shedding the light when I was stuck in dark corners. He always encourages me to take good care of both work and life, and was very supportive and considerate when I need to work from home to take care of the family. I can not say enough about how rewarding and fascinating to have Dan as an advisor.

Finally, I would like to thank my parents and my wife for all the support, care, happiness, love, and everything.

DEDICATION

To Jin

and

in loving memory of my dad

Chapter 1

INTRODUCTION

While compelling in the long term, Berners-Lee's vision of the Semantic Web [19] is developing slowly. Researchers have argued that the relative difficulty of authoring structured data is a primary cause [56]. A chicken-and-egg problem results: if there was more structured data available, people would develop applications; but without compelling applications, it is not worth people's time to structure their data. In order to break this deadlock, a bootstrapping method is needed — some method of automatically structuring a large amount of existing data.

The ideal vision is a machine reading system which autonomously extracts information from the Web. The system should strive to satisfy the following desiderata [89]:

High quality: the system should extract information with high accuracy;

Large scale: the system should acquire knowledge at Web-scale and be open to arbitrary domains, genres, and languages;

Maximal autonomy: the system should incur minimal human effort;

Total recall: the system should harvest both head and long tail textual knowledge.

These desiderata raise many intriguing and challenging research questions. The vast majority of information extraction work uses supervised learning of relation-specific examples. While these methods can achieve high precision and recall, they require too much human effort to scale. Instead, unsupervised or self-supervised techniques should be considered. Several systems of this form have been proposed, e.g. MULDER [66], AskMSR [24], KNOWITALL [51], TEXTRUNNER [14], and NELL [29], showing some signs of early success. The insight underlying these systems stems from the huge redundancy of knowledge on the Web — many things worth extracting are stated many times, in different ways and on

disparate Web pages. As a result, complex linguistic processing is unnecessary, because one of the occurrences is likely written in a form which can be correctly extracted with simple methods. Furthermore, the Web's statistical properties, as calculated by a search engine, are a powerful tool for extraction [36, 47, 49]. Unfortunately, many of the things published on the Web are incorrect (e.g. "Elvis killed John Kennedy"), and the increasing linguistic sophistication of link spam poses a growing challenge to these methods.

We propose a very different approach to massive information extraction. Instead of using the whole Web, we start from a single site: `en.wikipedia.org`, and use this as a first bootstrapping step to enable subsequent extraction from the Web as a whole.

1.1 Why Wikipedia

Focusing on Wikipedia largely solves the problem of inaccurate source data, but introduces new challenges. For example, redundancy is very greatly reduced — there is one single article for each unique concept in Wikipedia and much information is stated only once. This apparently increases the need for deep syntactic analysis. On the other hand, Wikipedia has several attributes that make it ideal for machine reading:

- Wikipedia gives all important concepts their own unique identifier — the URI of a definitional page. The first reference to such a concept typically includes a link which can be used for disambiguation. As a result, homonyms are much less of a problem than in unstructured text.

- Infoboxes are tabular summaries of an object's key attributes. Figure 1.1 shows a sample infobox from the article on "Beijing", which is generated using the template of the "Settlement" class. We model an infobox class as a relation schema C (e.g. *Settlement*) with a set of attributes $A(C) = \{C.a_i(e.g. Settlement.population)\}$.

- A *list* page shows a collection of instances for a class. For example, the "List of cities in China" page contains cities like "Beijing" and "Shanghai". We model a list page as a instance set for a class $I(C) = \{e_i | e_i \in C\}$, where e_i is an entity contained in the list page for class C.

- A Wikipedia article usually contains a *category* section, which provides membership information for the article's primary entity. For example, the categories for "Beijing" include "Capitals in Asia" and "Host cities of the Summer Olympic Games". We model the categories as a set of types for the entity $T(e) = \{t_i | e \in t_i\}$, where t_i is a category tag in the article for e.

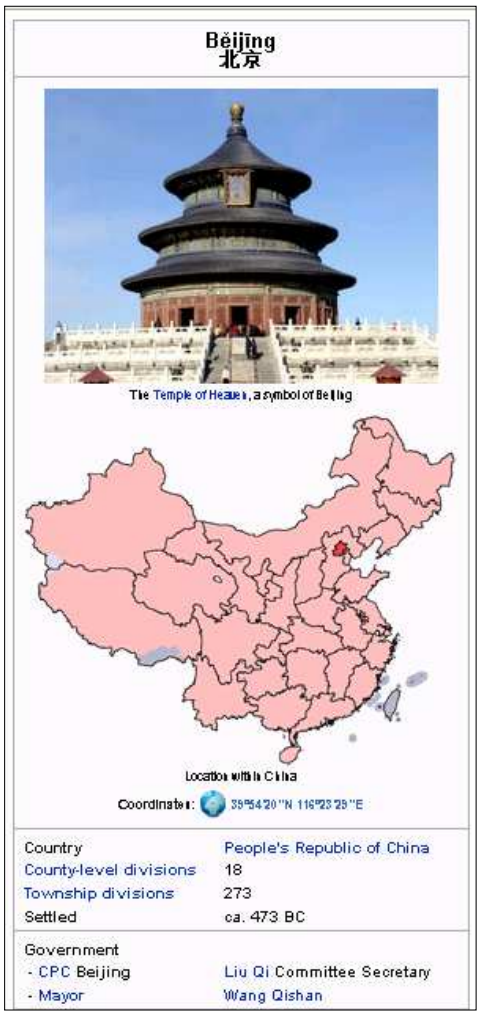
- A *redirection* page redirects a pointer for one term to another article (e.g., "Peking" is redirected to "Beijing"). We model all redirection pages to the same entity e as a synonym set $S_r(e) = \{e_i | e_i \sim e\}$, where e_i is redirected to e.

- A *disambiguation* page provides alternative meanings for one term. For example, the disambiguation page about "Jaguar" contains explanations for both the large cat and the luxury car. We abstract all disambiguation pages mentioning the same entity e into another kind of synonym set $S_d(e) = \{e_i | e_i \sim e\}$, where e_i is a disambiguation page containing a pointer to e.

- With over 3.3 million articles[1], Wikipedia is appropriately sized — big enough to provide a sufficient dataset, yet enough smaller than the full Web that a hundred-node cluster is unnecessary for corpus processing.

Using these special semantics in Wikipedia, we develop three systems to perform machine reading tasks: KYLIN, which exploits Wikipedia infoboxes to automatically create training examples to learn relation-specific extractors; KOG, which automatically generates Wikipedia Infobox Ontology by integrating heterogeneous evidence via joint inference using Markov Logic Networks; WOE, which learns unlexicalized open extractors by first matching Wikipedia infoboxes with sentences, then abstracting the matched sentences to relation-independent training examples. We also propose techniques to generalize these systems for bootstrapping toward the general Web. The following sections provide a summary of the main work of this dissertation.

[1]As measured in May 2010 for the English version of Wikipedia.




```
{{Infobox Settlement
|official_name  = Běijīng
|other_name  =
|native_name  = 北京
|settlement_type = [[Municipality of
                  China|Municipality
|image_skyline = SA Temple of Heaven.jpg
|image_caption  = The [[Temple of
          Heaven]], a symbol of Beijing
|citylogo_size  =
|image_map           = China-Beijing.png
|mapsize      = 275px
|map_caption         = Location within China
|subdivision_type     = Country
|subdivision_name     = [[People's Republic of
                  China]]
|subdivision_type1 = [[Political divisions of
                  China#County level|County-
                  level divisions]]
|subdivision_name1      = 18
|subdivision_type2  = [[Political divisions of
                  China#Township
          level|Township divisions]]
|subdivision_name2      = 273
|leader_title         =[[Communist Party of
                  China|CPC]] Beijing
|leader_name  =[[Liu Qi (Communist)|Liu Qi]]
                  Committee Secretary
|leader_title1        =  [[Mayor]]
|leader_name1         =[[Wang Qishan]]
|established_title     = Settled
|established_date      = ca. 473 BC
...
}}
```

Figure 1.1: Sample Wikipedia infobox and the attribute / value data used to generate it.

1.2 KYLIN: Relation-Specific Information Extraction Using Wikipedia

Our KYLIN system [108], introduced in Chapter 2, uses Wikipedia infoboxes to automatically create training examples to learn relation-specific extractors. For each attribute value in an infobox, KYLIN seeks a *unique* sentence in the article to match the attribute value. In this way, KYLIN automatically creates millions of training examples for thousands of relations. It then learns two classifiers to predict whether an article belongs to a target class (e.g. *Settlement*), and whether a sentence describes a target relation (e.g. *Settle-*

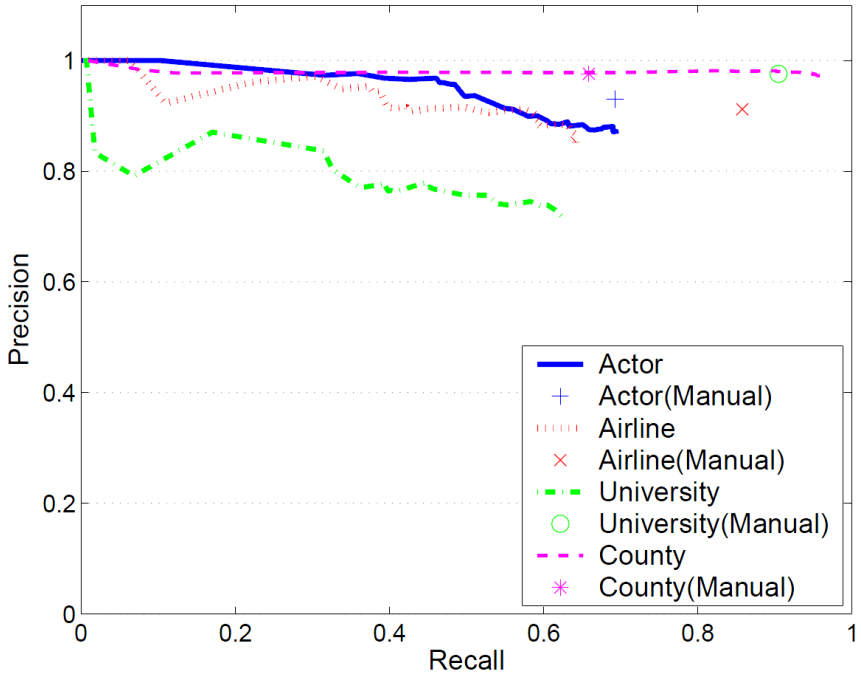


Figure 1.2: KYLIN achieves roughly comparable performance with human editors. For the "U.S. County" class it does even better. The individual points correspond to the performance of Wikipedia users' manual edition.

ment.population). It also learns conditional random fields (CRF) extractors to crop the detailed attribute values out of sentences.

Our experiments show that KYLIN's precision ranges from mid-70s to high-90s percent, depending on the attribute type and infobox class. Figure 1.2 shows its P/R curves on four sample infobox classes. We can see that KYLIN achieves roughly comparable performance with human editors; in one case, the "U.S. County" class, it does even better.

As far as we know, KYLIN is the first system that autonomously transfers knowledge from random editors' effort of collaboratively editing Wikipedia to learn extractors for thousands of relations. The Wikipedia heuristic of matching infoboxes with sentences to automatically label training examples is especially novel, enabling us to develop large-scale self-supervised learning systems. This heuristic can be further generalized to matching arbitrary structured KB with documents, as shown in [78, 107], where more KB (like Freebase,

DBpedia and IMDB) are used to match Web documents for generating labeled training examples to learn extractors. Similar approaches were also used in [17, 55], where bibliographic KB, like DBLP, are used to match research paper citation records in Web documents to generate labeled training examples. However, the bibliography domain only involves a few relations like *author, title, venue,* and *year*, and the research paper citation strings are semi-structured. In contrast, KYLIN aims to handle thousands of relations and totally unstructured texts.

1.3 KOG: Automatic Wikipedia Infobox Ontology Generation

In order to effectively exploit extracted data, the tuples must be organized using a clean and consistent ontology. Unfortunately, while Wikipedia has a category system for articles, the facility is noisy, redundant, incomplete, inconsistent and of very limited value for our purposes. Better taxonomies exist, of course, such as WordNet [1], but these don't have the rich attribute structure found in Wikipedia. To address this challenge, we built KOG, a system described in Chapter 3 that automatically builds a rich ontology by combining Wikipedia infoboxes with WordNet. KOG treats each infobox template as a class, and the slots of the template as attributes. It then performs three tasks. First, it cleans the infobox schemata by detecting duplicate classes and attributes, pruning ill-defined ones, recovering terse names (e.g., from "ABL" to "Australian Baseball League"), and inferring type information of attributes. Secondly, it predicts subsumption relationships between infobox classes. KOG computes six different kinds of features, some metric and some Boolean, and applies both support-vector machines (SVM) and Markov Logic Networks (MLNs) for ISA-relationship classification. The MLNs model is especially novel, simultaneously constructing a subsumption lattice and a mapping to WordNet using joint inference. The result for subsumption detection is shown in Figure 1.3, demonstrating the superiority of the joint inference approach. Finally, KOG maps attributes between related classes with estimated precision of 94% and recall of 87%, allowing property inheritance.

We demonstrate how the resulting ontology may be used to enhance Wikipedia in many ways, such as advanced query processing for Wikipedia facts, facetted browsing, automated infobox edits and template generation. It can benefit many other applications, such as

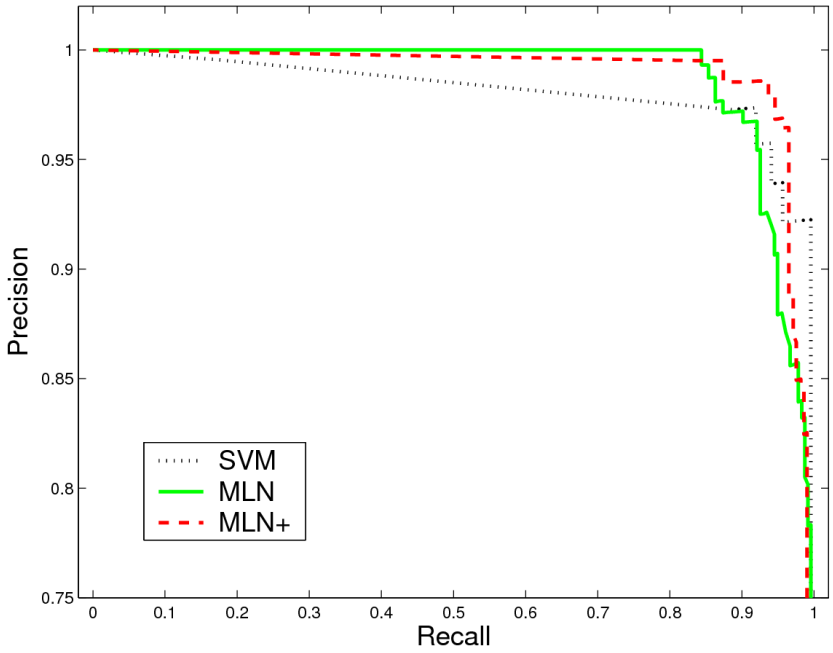


Figure 1.3: Subsumption detection via joint inference using MLNs is superior than applying a SVM model.

improving extractors with shrinkage, as we show in Chapter 4.

1.4 Moving Down the Long Tail

KYLIN works extremely well for popular infobox classes where users have previously created sufficient infoboxes to train an effective extractor model. For example, in the "U.S. County" class KYLIN has 97.3% precision with 95.9% recall. Unfortunately, however, many classes (e.g. "Irish Newspaper") contain only a *small number* of infobox-containing articles. For classes sitting on this long tail, KYLIN can't get enough training data — hence its extraction performance is often unsatisfactory for these classes.

Furthermore, even when KYLIN does learn an effective extractor there are numerous cases where Wikipedia has an article on a topic, but the article simply doesn't have much information to be extracted. Indeed, another long-tailed distribution governs the *length* of articles in Wikipedia; more than 40% Wikipedia articles are marked as *stub* pages, indicating that much-needed information is missing.

To meet the *total recall* desiderata of machine reading, we must confront the problems entailed by these long-tailed distributions in order to create a comprehensive semantic knowledge base summarizing the topics in Wikipedia. We must train extractors to operate on sparsely populated infobox classes and we must resort to other information sources if a Wikipedia article is superficial.

In Chapter 4 we describe three novel approaches for helping conquer the long-tailed challenges. First, we apply *shrinkage* [73, 102] when training an extractor of an instance-sparse infobox class by aggregating data from its parent and children classes. For example, knowing that *performer ISA person*, and *performer.location=person.birth_place*, we can use values from *person.birth_place* to help train an extractor for *performer.location*. We use the ontology learned by KOG to provide the subsumption hierarchy. This shrinkage technique improves recall by a factor of between 0.57 and 4.6. Secondly, we map the contents of known Wikipedia infobox data to TextRunner, a state-of-the-art open information extraction system [15]. This enables KYLIN to clean and augment its training dataset. When applied in conjunction with shrinkage, this *retraining* technique improves recall by a factor of between 1.4 and 5.9. Finally, when it is unable to extract necessary information from a Wikipedia page, we enable KYLIN to retrieve relevant sentences from the greater Web for extraction. Our techniques work best in concert. Together, they improve recall by a factor of 1.89 to 8.42 while maintaining or increasing precision. Figure 1.4 shows the effects of applying these techniques to the "performer" class. In addition to showing the great cumulative effect of these techniques, we analyze several variations of each method, exposing important engineering tradeoffs.

1.5 WOE: Open Information Extraction Using Wikipedia

KYLIN can automatically learn thousands of extractors for the relations defined in Wikipedia infoboxes. However, compared with an unbounded number of relations embedded in Web documents, this is still very limited. We build WOE, a system introduced in Chapter 5 that learns an open information extractor using Wikipedia. Specifically, WOE generates *relation-specific* training examples by matching infobox attribute values to corresponding sentences (as done in KYLIN), but WOE abstracts these examples to *relation-independent*

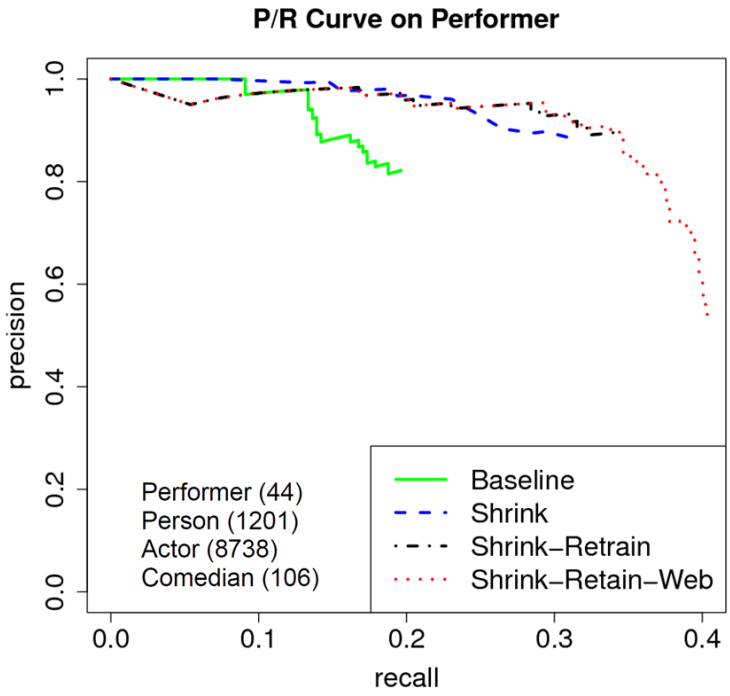


Figure 1.4: Combining KYLIN's extractions from Wikipedia and the Web yields a substantial improvement in recall without compromising precision. Already, shrinkage and retraining improved recall over the original KYLIN system, here the baseline, but the combination of extractions from Wikipedia and the Web, shrink-retrain-Web, performs even better.

training data to learn an unlexicalized extractor, akin to that of TextRunner. WOE can operate in two modes: when restricted to shallow features like part-of-speech (POS) tags, it learns a second-order linear chain CRF extractor and runs as quickly as Textrunner — 0.022 seconds per sentence in average; but when set to use parse features, it learns a pattern classifier based on shortest-dependency-path features whose precision and recall rise even higher. However, this performance improvement comes at the cost of speed: it takes 0.679 seconds to process one sentence — 30X times slower.

When restricted to shallow features, WOE yields an F-measure between 0.447 and 0.471 (*i.e.* between 9% and 23% greater than that of TextRunner) on three corpora; when set to use parse features, WOE achieves an F-measure between 0.572 and 0.650 (*i.e.* between 51% and 70% higher than that of TextRunner). Figure 1.5 shows the P/R curves of

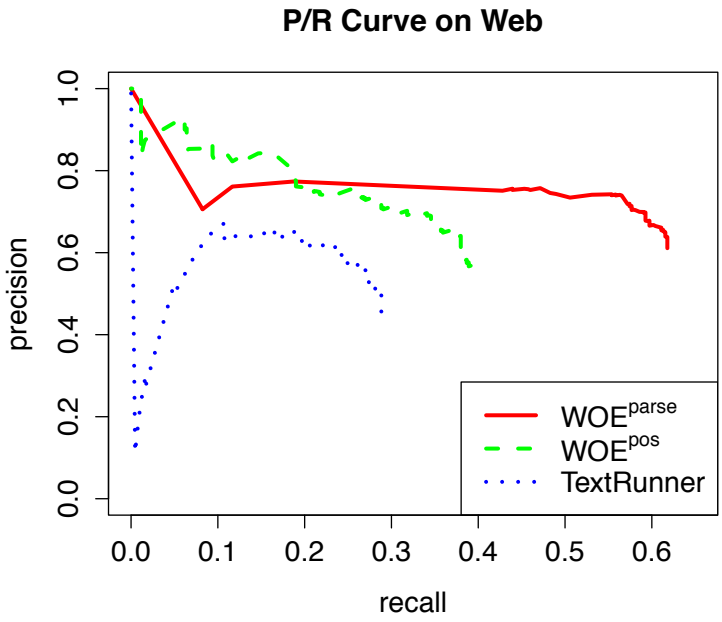


Figure 1.5: WOE^{pos} performs better than TextRunner, especially on precision. WOE^{parse} dramatically improves performance, especially on recall.

different extractors on a randomly sampled Web corpus with 300 sentences. Our extensive experiments uncover two sources of WOE's strong performance: the Wikipedia heuristic is responsible for the bulk of WOE's improved accuracy; dependency-parse features are highly informative when performing unlexicalized extraction, which is a different conclusion from previous evidence in [63].

1.6 Contributions

This dissertation embodies several important contributions:

- We propose bootstrapping from Wikipedia towards Web-scale machine reading and identify some unique challenges (lack of redundancy) and opportunities (unique identifiers, user-supplied training data, lists, categories, etc.) of this approach. We also identify additional issues resulting from Wikipedia's growth through decentralized authoring (e.g., inconsistency, schema drift, etc.). This high-level analysis should benefit

future work on Wikipedia and similar collaborative knowledge repositories.

- We show that matching structured records in a KB (e.g. Wikipedia infoboxes) with unstructured text (e.g. Wikipedia articles) can automatically label a large amount of training examples, which enable us to develop large-scale self-supervised learning systems. The matching process is surprisingly hard and we apply several heuristics to ensure high-quality labeled examples.

- We develop three systems (KYLIN, KOG and WOE) using Wikipedia via self-supervised learning. Collaboratively authored data is rife with noise and incompleteness. We identify robust learning methods which can cope in this environment. The results in our experiments show that focusing on Wikipedia enables us to learn high performance machine reading systems, which provides a solid base to bootstrap toward the general Web.

- We demonstrate that parser-based features are highly informative when performing unlexicalized extraction, which is a different conclusion from previous work in [63], where they assumed the presence of lexical features.

In the following chapters, we present each of our systems in more details. These projects make substantial steps toward addressing the challenges of machine reading. They also point to interesting future work involving extensions to each system as well as new overall approaches.

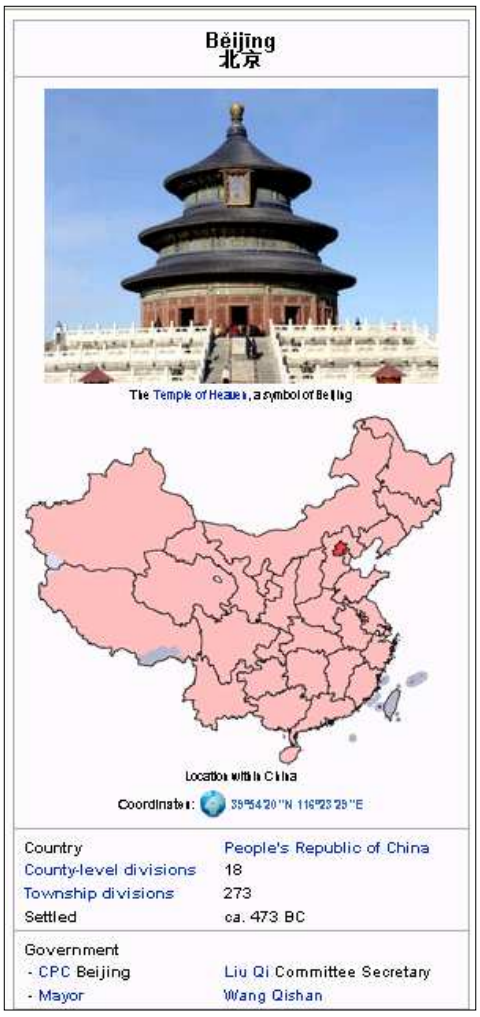
Chapter 2

KYLIN: RELATION-SPECIFIC INFORMATION EXTRACTION USING WIKIPEDIA

We are motivated by a vision of self-supervised machine reading — a system which can autonomously distill and organize semantic data from natural-language texts on the World Wide Web. Such a system could be useful for next-generation information retrieval, question answering and much more. Autonomy is crucial, since the scale of available knowledge is vast. We share this vision with a number of other projects, such as Snowball [9], KnowItAll [51], Mulder [66], AskMSR [24], TextRunner [14], and NELL [29]. The insight underlying these systems stems from the huge redundancy of knowledge on the Web — many things worth extracting are stated many times, in different ways and on disparate Web pages. As a result, complex linguistic processing is unnecessary, because one of the occurrences is likely written in a form which can be correctly extracted with simple methods. Furthermore, the Web's statistical properties, as calculated by a search engine, are a powerful tool for extraction [36, 47, 49]. Unfortunately, many of the things published on the Web are incorrect (e.g. "Elvis killed John Kennedy"), and the increasing linguistic sophistication of link spam poses a growing challenge to these methods.

We propose a very different approach to massive information extraction. Instead of using the whole Web, we argue that Wikipedia is an important focus to start from. If we can render much of Wikipedia into semantic form, it will be much easier to bootstrap toward the Web from that base.

Focusing on Wikipedia largely solves the problem of inaccurate source data, but introduces new challenges. For example, redundancy is very greatly reduced. On the other hand, Wikipedia has several attributes (unique identifiers, user-supplied training data, lists, categories, etc.) that make it ideal for machine reading. We develop the KYLIN system which performs relation-specific IE using Wikipedia.




```
{{Infobox Settlement
|official_name  = Běijīng
|other_name  =
|native_name  = 北京
|settlement_type = [[Municipality of
                   China|Municipality
|image_skyline = SA Temple of Heaven.jpg
|image_caption  = The [[Temple of
          Heaven]], a symbol of Beijing
|citylogo_size  =
|image_map           = China-Beijing.png
|mapsize      = 275px
|map_caption          = Location within China
|subdivision_type     = Country
|subdivision_name     = [[People's Republic of
                   China]]
|subdivision_type1 = [[Political divisions of
                   China#County level|County-
                   level divisions]]
|subdivision_name1     = 18
|subdivision_type2  = [[Political divisions of
                   China#Township
          level|Township divisions]]
|subdivision_name2     = 273
|leader_title         =[[Communist Party of
                   China|CPC]] Beijing
|leader_name  =[[Liu Qi (Communist)|Liu Qi]]
                   Committee Secretary
|leader_title1        =  [[Mayor]]
|leader_name1         =[[Wang Qishan]]
|established_title     = Settled
|established_date      = ca. 473 BC
…
}}
```

Figure 2.1: Sample Wikipedia infobox and the attribute / value data used to generate it.

2.1 Motivation Application: Infobox Completion

The motivation application behind KYLIN is *automatic infobox completion*. Many Wikipedia articles include *infoboxes*, a concise, tabular summary of the subject's attributes. Figure 2.1 shows a sample infobox from the article on "Beijing". We model an infobox class as a relation schema C (e.g. *Settlement*) with a set of attributes $A(C) = \{C.a_i(e.g. Settlement.population)\}$.

Because of their relational nature, infoboxes may be easily converted to semantic form as shown by Auer and Lehmann's DBpedia [12]. Furthermore, for each class of objects,

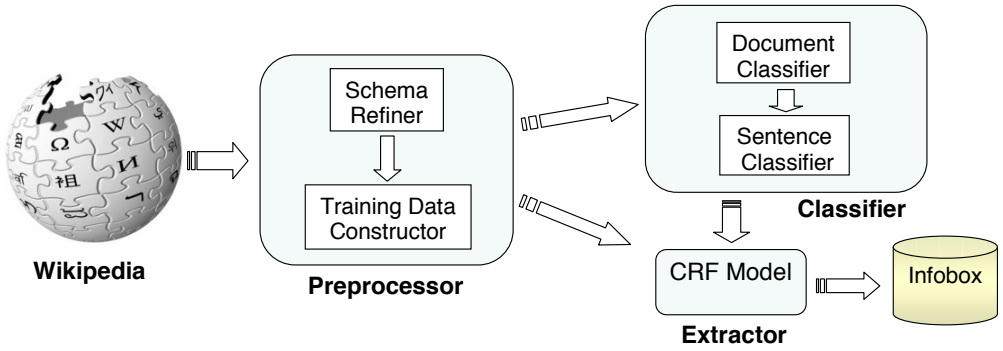


Figure 2.2: Architecture of KYLIN.

infoboxes and their templates implicitly define the most important and representative attributes; hence, infoboxes are valuable ontological resources. KYLIN automatically constructs and completes infoboxes with information distilled from unstructured Wikipedia articles. The basic idea is to use existing infoboxes as a source of training data with which to learn extractors for gathering more data. As shown in Figure 2.2, KYLIN has three main components: preprocessor, classifier, and extractor.

The *preprocessor* performs several functions. First, it selects and refines infobox schemata, choosing relevant attributes. Secondly, the preprocessor generates a dataset for training machine learners.

KYLIN trains two types of *classifiers*. The first type predicts whether a given Wikipedia article belongs to certain class C. The second type of classifier predicts whether a given sentence contains the value of a given attribute $C.a_i$. If there are C classes, KYLIN automatically learns $2C$ different classifiers.

KYLIN learns one extractor for each $C.a_i$ (per attribute per class), which identifies and clips out the necessary attribute value for $C.a_i$ from a given sentence s. Each extractor is a conditional random fields (CRF) model. Training data are taken from existing infoboxes as dictated by the predictions of the classifiers.

We explain the operation and performance of these components in the next section. But first we discuss the nature of the existing Wikipedia infoboxes and why they are harder to use for training than might be expected.

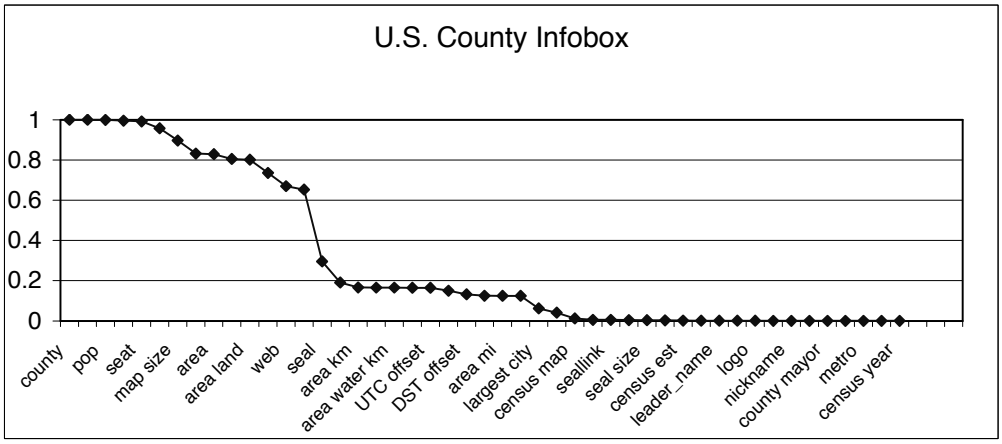


Figure 2.3: Usage percentage for attributes of the "U.S. County" infobox template shows that many attributes are used very rarely.

2.1.1 Challenges for Infobox Completion

While infoboxes contain much valuable information, they suffer from several challenging problems:

Incompleteness: Since infobox and article text are kept separate in Wikipedia, existing infoboxes are manually created when human authors create or edit an article — a tedious and time-consuming process. As a result, many articles have no infoboxes and the majority of infoboxes which *do* exist are incomplete. Still there in many classes, there is plenty of data for training.

Inconsistency: The manual creation process is noisy, causing contradictions between the article text and the infobox summary. For example, when we manually checked a random sample of 50 infoboxes in the "U.S. County" class, we found that 16% contained one or more errors[1]. We suspect that many of the errors are introduced when an author updates an article with a revised attribute value (e.g. population) and neglects to change both the text and the infobox — another effect of keeping infobox and text separate.

---

[1]Unless noted otherwise, all statistics in this chapter are taken from the 02/06/2007 snapshot of Wikipedia's English language version.

Schema Drift: Since users are free to create or modify infobox templates, and since they typically create an article by copying parts (e.g. the infobox template) from a similar article, the infobox schema for a class of articles tends to evolve during the course of authoring. This leads to several problems: schema duplication, attribute duplication, and sparseness. As an example of schema duplication, note that four different templates: "U.S. County" (1428), "US County" (574),"Counties"(50) and "County" (19) are used to describe the same type of object. Similarly, multiple tags denote the same semantic attribute. For example, "Census Yr", "Census Estimate Yr", "Census Est." and "Census Year" all mean the same thing. Furthermore, many attributes are used very rarely. Figure 2.3 shows the percent usage for the attributes of the "U.S. County" infobox template; only 29% of the attributes are used by 30% or more of the articles, and only 46% of the attributes are used by at least 15% of the articles.

Typefree System: The design of Wikipedia is deliberately low-tech, to facilitate human generation of content, and infoboxes are no exception. In particular, there is no type system for infobox attributes. For example, the infobox for "King County, Washington" has a tuple binding the attribute "land area" to equal "2126 square miles" and another tuple defining "land area km" to be "5506 square km" despite the fact that one can be easily derived from another. Clearly this simple approach bloats the schema and increases inconsistency; the similarity between these related attributes also increases the complexity of extraction.

Irregular Lists: List pages, which link to large numbers of similar articles, are a potential source of valuable type information. Unfortunately, because they are designed for human consumption, automated processing is difficult. For example, some list pages separate information in items, while others use tables with different schemas. Sometimes, lists are nested in an irregular, hierarchical manner, which greatly complicates extraction. For example, the "List of cities, towns, and villages in the United States" has an item called "Places in Florida", which in turn contains "List of counties in Florida."

Flattened Categories: While Wikipedia's category tag system seems promising as a source of ontological structure, it is so flat and quirky that utility is low. Furthermore, many tags are purely administrative, e.g. "Articles to be merged since March 2007."

2.2 KYLIN: Relation-Specific Information Extraction

KYLIN uses existing infoboxes as a source of training data to learn extractors for distilling information from Wikipedia articles. The input to KYLIN is the raw Wikipedia corpus, and the output is a collection of extractors for a broad set of relations found in Wikipedia infoboxes. Figure 2.4 shows the compact pseudocode for KYLIN. We discuss each component of KYLIN in more details in the following sections.

2.2.1 Preprocessor

The preprocessor is responsible for creating a training suite that can be used to learn extraction code for creating infoboxes. The input to the preprocessor is the raw Wikipedia corpus; the outputs include a cleaned attribute set $\{C.a_i\}$ for each infobox class C, and a matching sentence set for each $C.a_i$, which can be used as training dataset to train classifiers and extractors by the subsequent modules. The preprocessor performs two tasks: schema refinement and training dataset construction.

Schema Refinement: The previous section explained how collaborative authoring leads to infobox schema drift, resulting in the problems of schema duplication, attribute duplication and sparsity. Thus, a necessary prerequisite for generating good infoboxes for a given class is determining a uniform target schema.

This can be viewed as an instance of the difficult problem of schema matching [45]. Clearly, many sophisticated techniques can be brought to bear, but we adopt a simple statistical approach for our prototype. KYLIN scans the Wikipedia corpus and selects all infobox classes that are used frequently enough (e.g., by more than 5 instance articles). For each selected infobox class, KYLIN retrieves all instance articles containing the class, catalogs all attributes mentioned and selects the most common. Our current implementation restricts attention to attributes used in at least 15% of the articles, which yields plenty of training data.

Constructing Training Datasets: Next, the preprocessor constructs training datasets for use when learning classifiers and extractors. KYLIN iterates through the articles. For each article with an infobox mentioning one or more target attributes, KYLIN segments the

KYLIN (Wikipedia Corpus):

1. Preprocess Corpus:

1.1 Refine Infobox Shemata:

IB = {}

For each infobox class C

If C is used by few instance articles

Continue

A(C) = {}

For each attribute $C.a_i$ of C

If $C.a_i$ is used by a large portion of instance articles of C

Add $C.a_i$ to A(C)

Add <C, A(C)> to IB

1.2 Construct Training Dataset by Matching Infoboxes with Sentences:

TD = {}

For each <C, A(C)> in IB

Get all instance articles of C: D(C) = {d|d contains C }

For each d in D(C)

For each $C.a_i$ which has an attribute value in d's infobox

s = matching sentence in d that mentions the attribute value of $C.a_i$

Add <C, $C.a_i$, s> to TD

2. Learn Classifiers:

2.1 Train Document Classifier:

CL_d = {}

For each C

Learn a document classifier CL_d(C) based on list pages and category tags

Add CL_d(C) to CL_d

2.2 Train Sentence Classifier:

CL_s = {}

For each C

Learn a multi-class Maximum Entropy sentence classifier CL_s(C) based on TD

Add CL_s(C) to CL_s

3. Learn Extractors:

EX = {}

For each $C.a_i$

Learn a CRF extractor EX($C.a_i$) based on TD and add it to EX

Return IB, CL_d, CL_s, EX

Figure 2.4: The KYLIN system matches Wikipedia infboxes with sentences to create training examples, and learns a broad set of relation-specific extractors.

document into sentences, using the OpenNLP library [2]. Then, for each target attribute, KYLIN tries to find a unique, corresponding sentence in the article. The resulting labelled sentences form positive training examples for each attribute. Other sentences form negative training examples.

Our current implementation uses several heuristics to match sentences to attributes. If an attribute's value is composed of several sub-values (e.g. "hub cities"), KYLIN splits them and processes each sub-value as follows:

1. For each internal hyperlink in the article and the infobox attributes, find its unique primary URI in Wikipedia (through a redirect page if necessary). For example, both "USA" and "United States of America" will be redirected to "United States". Replace the anchor text of the hyperlink with this identifier.

2. If the attribute value is mentioned by exactly one sentence in the article, use that sentence and the matching token as a training example.

3. If the value is mentioned by several sentences, KYLIN determines what percentage of the tokens in the attribute's *name* are in each sentence. If the sentence matching the highest percentage of tokens has at least 60% of these keywords, then it is selected as a positive training example. For example, if the current attribute is "TotalArea: 123", then KYLIN might select the sentence "It has a total area of 123 square kms", because "123" and the tokens "total" and "area" are all contained in the sentence.

Unfortunately, there are several difficulties preventing us from getting a perfect training dataset. First, OpenNLP's sentence detector is imperfect. Second, the article may not even have a sentence which corresponds to an infobox attribute value. Third, we require exact value-matching between attribute values in the sentence and infobox. While this strict heuristic ensures precision, it substantially lowers recall. The values given in many are incomplete or written differently than in the infobox. Together, these factors conspire to produce a rather incomplete dataset.[2] Fortunately, we are still able to train our learning algorithms effectively.

[2]Alternatively, one can view our heuristics as explicitly preferring incompleteness over noise — a logical consequence of our choice of high precision extraction over high-recall.

2.2.2 Classifier

The inputs to the classifier module include the training dataset created by the preprocessor, and list pages and category tags in Wikipedia; the outputs are two types of classifiers — for each class of article being processed, a heuristic *document classifier* is used to recognize members of the class, a multi-class Maximum Entropy *sentence classifier* is trained in order to predict whether a given sentence is likely to contain certain attributes' values.

Document Classifier: To accomplish autonomous infobox generation, KYLIN must first locate candidate articles for a given class — a familiar document classification problem. Wikipedia's manually-generated list pages, which gather concepts with similar properties, and category tags are highly informative features for this task. For example, the "List of U.S. counties in alphabetical order" points to 3099 items; furthermore, 68% of those items have additionally been tagged as "county" or "counties." Eventually, we will use lists and tags as features in a Naive Bayes, Maximum Entropy or SVM classifier, but as an initial baseline we used a simple, heuristic approach. First, KYLIN locates all list pages whose titles contain infobox class keywords. Second, KYLIN iterates through each page, retrieving the corresponding articles but ignoring tables. If the category tags of the retrieved article also contains infobox class keywords, KYLIN classifies the article as a member of the class. As shown in Section 2.3.1, our baseline document classifier achieves very high precision (98.5%) and reasonable recall (68.8%).

Sentence Classifier: It proves useful for KYLIN to be able to predict which attribute values, if any, are contained in a given sentence. This can be seen as a multi-class, multi-label, text classification problem. To learn these classifiers, KYLIN uses the training set produced by the preprocessor (Section 2.2.1). For features, we seek a domain-independent set which is fast to compute; our current implementation uses the sentence's tokens and their part of speech (POS) tags as features.

For our classifier, we employed a Maximum Entropy model [81] as implemented in Mallet [74], which predicts attribute labels in a probabilistic way — suitable for multi-class and multi-label classifications.[3] To decrease the impact of a noisy and incomplete training

[3]We also experimented with a Naive Bayes model, but its performance was slightly worse.

dataset, we employed bagging [23] rather than boosting [76] as recommended by [82].

2.2.3 Extractor

The input to the extractor module is the training dataset created by the preprocessor, and the output is a set of extractors — one for each infobox attribute $C.a_i$. Extracting attribute values from a sentence may be viewed as a sequential data-labeling problem. We use the features shown in Table 2.1. Conditional random fields (CRFs) [67] are a natural choice given their leading performance on this task; we use the Mallet [74] implementation. We were confronted with two interesting choices in extractor design, and both concerned the role of the sentence classifier. We also discuss the issue of multiple extractions.

Training Methodology: Recall that when producing training data for extractor-learning, the preprocessor uses a strict pairing model. Since this may cause numerous sentences to be incorrectly labelled as negative examples, KYLIN uses the sentence classifier to *relabel* some of the training data as follows. All sentences which were assigned to be negative training examples by the preprocessor are sent through the sentence classifier; if the classifier disagrees with the preprocessor (*i.e.*, it labels them positive), then they are eliminated from the training set for this attribute. Experiments in Section 2.3.3 show that this small adjustment greatly improves the performance of the learned CRF extractor.

KYLIN trains a different CRF extractor for each attribute, rather than training a single master extractor that clips all attributes. We chose this architecture largely for simplicity — by keeping each attribute's extractor independent, we ensure that the complexity does not multiply.

Classifier's Role in Extraction: We considered two different ways to combine the sentence classifier and extractor for infobox generation. The first is an intuitive pipeline mode where the sentence classifier selects the sentences which should be sent to the CRF extractor. We expected that this approach would decrease the number of false positives with a potential loss in recall. The second architecture treats the classifier's prediction as a CRF feature, but applies the extractor to all sentences. We expected better recall at the expense of speed. The experiments of Section 2.3.3 shows that our expectations were

Feature Description	**Example**
First token of sentence	*Hello* world
In first half of sentence	*Hello* world
In second half of sentence	Hello *world*
Start with capital	Hawaii
Start with capital, end with period	Mr.
Single capital	A
All capital, end with period	CORP.
Contains at least one digit	AB3
Made up of two digits	99
Made up of four digits	1999
Contains a dollar sign	20$
Contains an underline symbol	km_square
Contains an percentage symbol	20%
Stop word	the; a; of
Purely numeric	1929
Number type	1932; 1,234; 5.6
Part of Speech tag	
Token itself	
NP chunking tag	
String normalization: capital to "A", lowercase to "a", digit to "1", others to "0"	$TF-1 \Longrightarrow AA01$
Part of anchor text	Machine *Learning*
Beginning of anchor text	*Machine* Learning
Previous tokens (window size 5)	
Following tokens (window size 5)	
Previous token anchored	Machine *Learning*
Next token anchored	*Machine* Learning

Table 2.1: Feature sets used by the CRF extractor

fulfilled, but the pipeline's boost to precision was higher than expected, creating a more effective architecture.

Multiple Extractions: Sometimes the extractor finds multiple values for a single attribute. This often happens as a mistake (e.g. because of an extractor error or redundant text in the article) but can also happen when the attribute is not functional (e.g. a band likely has several members). KYLIN distinguishes the cases by seeing if multiple values are found in the attribute's training set. If so, the set of extractions is returned as the final result. Otherwise, KYLIN returns the single value with the highest confidence.

2.3 Experiments

To avoid overloading the Wikipedia server, we downloaded the 2007.02.06 data for testing. We selected four popular classes: U.S. county, airline, actor, and university. Each was among the top 100 classes in terms of infobox usage. We address four questions:

- What are the precision and recall of the document classifier?
- What are the precision and recall of the infobox attribute extractor and how does it compare to the performance of human users?
- Is the precision of the extractor improved by pruning the set of training data with the sentence classifier? What is the cost in terms of recall?
- Should one use the sentence classifier as a serial pipeline filter preceding the extractor or simply make the classifier's output available as a feature for the extractor's use?

2.3.1 Document Classifier

We use sampling plus human labelling to estimate the precision and recall of the classifiers. We measure the precision of a class' classifier by taking a random sample of 50 pages which were predicted to be of that class and manually checking their accuracy. Table 2.2 lists the estimated precision for our four classes. On average, the classifiers achieve 98.5% precision.

	County	Airline	Actor	University
Predicted	3302	2764	6984	4309
Precision (%)	100	100	100	94

Table 2.2: Estimated precision of the document classifier.

To estimate recall, we introduce some notation, saying that an article is *tagged* with a class if it has had an infobox of that type manually created by a human author. We use the set of tagged pages as a sample from the universal set and count how many of them are identified by the classifier. Table 2.3 shows the detailed results, but averaging uniformly over the four classes yields an average recall of 68.8%. This is quite good for our baseline implementation, and it seems likely that a machine-learning approach could result in substantially higher recall.

	County	Airline	Actor	University
Tagged	1245	791	3819	4025
Recall (%)	98.1	85.3	41.3	50.3

Table 2.3: Estimated recall of the document classifier

Note that there are some potential biases which might potentially affect our estimates of precision and recall. First, as mentioned in Section 2.1.1, some list pages are challenging to exploit, and list page formatting varies a lot between different classes. Second, articles with user-added infobox classes tend to be on more popular topics, and these may have a greater chance to be included in list pages. This could lead to minor overestimation of KYLIN recall. But we believe that these factors are small, and our estimates of precision and recall are accurate.

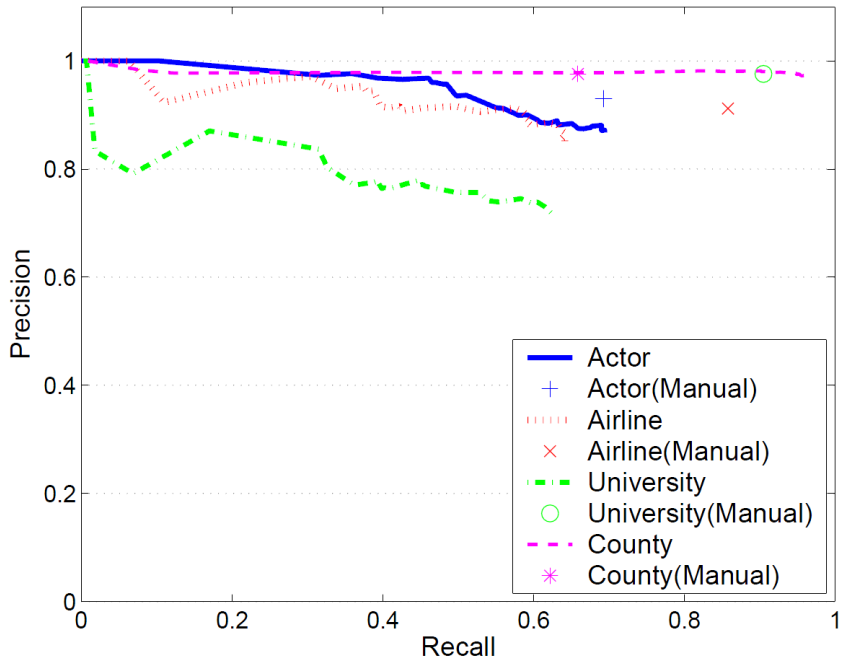


Figure 2.5: KYLIN achieves roughly comparable performance with human editors. For the "U.S. County" class it does even better. The individual points correspond to the performance of Wikipedia users' manual edition.

2.3.2 Extractor Performance

In order to be useful as an autonomous system, KYLIN must be able to extract attribute values with very high precision. High recall is also good, but of less importance. Since our CRF extractor outputs a confidence score for its extraction, we can modulate the confidence threshold to control the precision/recall tradeoff as shown in Figure 2.5.

Interestingly, the precision/recall curves are extremely flat, which means the precision is rather stable w.r.t the variation of recall. In practice, KYLIN is able to automatically tune the confidence threshold based on training data provided by the preprocessor for various precision/recall requirements. In order to reduce the need for human fact checking, one can set a high threshold (e.g. 0.99), boosting precision. A lower threshold (e.g. 0.3) extends recall substantially, at only a small cost in precision.

In our next experiment, we use a fixed threshold of 0.6, which achieves both reason-

able precision and recall for all classes. We now ask how KYLIN compares against strong competition: human authors. For each class, we randomly selected 50 articles with existing infobox templates. By manually extracting all attributes mentioned in the articles, we could check the performance of both the human authors and of KYLIN. The results are shown in Table 2.4. We were proud to see that KYLIN performs better on the "U.S. County" domain, mainly because its numeric attributes are relatively easy to extract. In this domain, KYLIN was able to successfully recover a number of values which had been neglected by humans. For the "Actor" and "Airline" domains, KYLIN performed slightly worse than people. And in the "University" domain, KYLIN performed rather badly, because of implicit references and the type of flexible language used in those articles. For example, KYLIN extracted "Dwight D. Eisenhower" as the president of "Columbia University" from the following sentence.

- *Former U.S. President Dwight D. Eisenhower served as President of the University.*

Unfortunately, this is incorrect, because Eisenhower was a *former* president (indicated somewhere else in the article) and thus the incorrect value for the current president.

Implicit expressions also lead to challenging extractions. For example, the article on "Binghamton University" individually describes the number of undergraduate and graduate students in each college and school. In order to correctly extract the total number of students, KYLIN would need to reason about disjoint sets and perform arithmetic, which is beyond the abilities of most textual entailment systems [42, 70], let alone one that scale to a Wikipedia-sized corpus.

2.3.3 Using the Sentence Classifier with the Attribute Extractor

Recall that KYLIN uses the sentence classifier to prune some of the negative training examples generated by the preprocessor before training the extractor. We also explored two ways of connecting the sentence classifier to the extractor: as a pipeline (where only sentences satisfying the classifier are sent to the extractor) or by feeding the extractor every sentence, but letting it use the classifier's output as a feature. In this experiment, we consider four possible configurations:

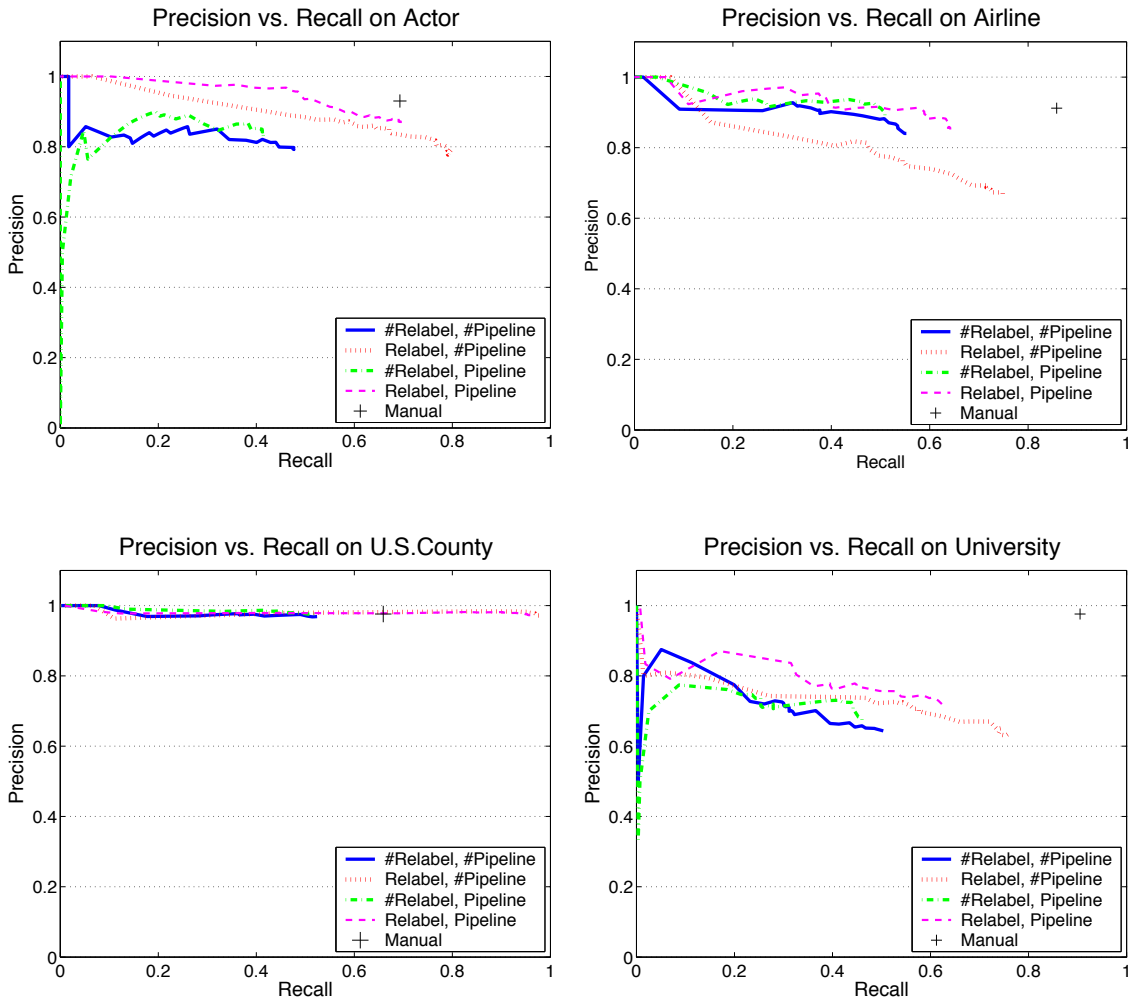


Figure 2.6: Using classifier to refine the training dataset helps to improve KYLIN's performance especially in terms of robustness and recall. The pipeline structure helps to improve precision at a modest cost of recall. The cross points correspond to Wikipedia users' manual edition.

Class	People		System	
	Pre.(%)	Rec.(%)	Pre.(%)	Rec.(%)
County	97.6	65.9	97.3	95.9
Airline	92.3	86.7	87.2	63.7
Actor	94.2	70.1	88.0	68.2
University	97.6	90.5	73.9	60.5

Table 2.4: Relative performance of people and KYLIN on infobox attribute extraction.

- Relabel, Pipeline — uses the classifier's results to relabel the training dataset for the extractor and uses a pipeline architecture.
- Relabel, #Pipeline — also uses the classifier's results to relabel the training dataset for the extractor, but doesn't use a pipeline (instead it provides the classifier's output to the CRF extractor).
- #Relabel, Pipeline — training examples are not relabelled, but the pipelined architecture is used.
- #Relabel, #Pipeline — training examples are not relabelled and a pipeline is eschewed (the classifier's output is fed directly to the extractor).

Figure 2.6 shows the detailed results. In most cases the "Relabel, Pipeline" policy achieves the best performance. We draw the following observations:

- Noise and incompleteness within the training dataset provided by the Preprocessor makes the CRF extractor unstable, and hampers its performance (especially recall) in most cases.
- By using classifier to refine the training dataset, many false negative training examples are pruned; this helps to enhance the CRF extractor's performance, especially in terms of robustness and recall.

- The pipeline architecture improves precision in most cases by reducing the risk of false positive extractions on irrelevant sentences. But since fewer sentences are even given to the extractor, recall suffers.

2.4 Related Work

We group related work into several categories: bootstrapping the semantic web, unsupervised and self-supervised information extraction, extraction from Wikipedia, and related Wikipedia-based systems.

Bootstrapping the Semantic Web: REVERE [56] aims to cross the chasm between structured and unstructured data by providing a platform to facilitate the authoring, querying and sharing of data. It relies on human effort to gain semantic data, while our KYLIN is fully autonomous. DeepMiner [110] bootstraps domain ontologies for semantic web services from source web sites. It extracts concepts and instances from semi-structured data over source interface and data pages, while KYLIN handles both semi-structured and unstructured data in Wikipedia. The SemTag and Seeker [44] systems perform automated semantic tagging of large corpora. They use the TAP knowledge base [94] as the standard ontology, and use it to match instances on the Web. In contrast, KYLIN doesn't assume any particular ontology, and tries to extract all desired semantic data within Wikipedia.

Unsupervised and Self-Supervised Information Extraction: Since the Web is large and highly heterogeneous, unsupervised and self-supervised learning is necessary for scaling. Several systems of this form have been proposed. MULDER [66] and AskMSR [24, 49] use the Web to answer questions, exploiting the fact that most important facts are stated multiple times in different ways, which licenses the use of simple syntactic processing. KNOWITALL [51] uses search engines to compute statistical properties enabling extraction. NELL [29] continuously learns many classifiers at once from primarily unlabeled data, coupling the learning of these classifiers in order to improve accuracy. Bunescu and Mooney use search engines to construct positive and negative sample sentences based on a few seeds for a given relation, and train a SVM extractor using string kernels, extending the MIL framework [27]. Each of these systems relies heavily on the Web's information redundancy.

However, unlike the Web, Wikipedia has little redundancy — there is only one article for each unique concept in Wikipedia. Instead of utilizing redundancy, KYLIN exploits Wikipedia's unique structure and the presence of user-tagged data to train machine learners. Mintz *et al.* [78] uses Freebase to provide distant supervision for relation extraction. They applied a similar heuristic by matching Freebase tuples with unstructured sentences (Wikipedia articles in their experiments) to create features for learning relation extractors. A similar approach is applied in [107] where more KB besides Freebase (like DBpedia, and IMDB) and a larger corpus are used to match sentences. Using outside KB to match arbitrary sentences instead of matching Wikipedia infobox within corresponding articles will potentially increase the size of matched sentences at a cost of accuracy. Similar approaches were also used in [17, 55], where bibliographic KB, like DBLP, are used to match research paper citation records in Web documents to generate labeled training examples. However, the bibliography domain only involves a few relations like *author, title, venue,* and *year*, and the research paper citation strings are semi-structured. In contrast, KYLIN deals with thousands of relations and totally unstructured texts.

Information Extraction from Wikipedia: Several other systems have addressed information extraction from Wikipedia. Nakayama *et al.* [79] parse selected Wikipedia sentences and perform extraction over the phrase structure trees based on several handcrafted patterns. Auer and Lehmann developed the DBpedia [12] system which extracts information from existing infoboxes within articles and encapsulate them in a semantic form for query. In contrast, KYLIN populates infoboxes with *new* attribute values. Suchanek *et al.* describe the YAGO system [103] which extends WordNet using facts extracted from Wikipedia's category tags. But in contrast to KYLIN, which can learn to extract values for *any* attribute, YAGO only extracts values for a limited number of predefined relations. Nguyen *et al.* proposed to extract relations from Wikipedia by exploiting syntactic and semantic information [80]. Their work is the most similar with ours in the sense of stepping towards autonomously semantifying both semi-structured and unstructured data. However, there are still several obvious distinctions. First, their system only classifies whether a sentence is related to some attribute, while KYLIN also *extracts* the particular attribute

value within the sentences. Second, they only care about the relationship-typed attributes between concepts (i.e. objects having their own identifying pages), while KYLIN tries to extract *all* important attributes. Third, their system targets a limited number of predefined attributes, while KYLIN can dynamically refine infobox templates for different domains. Recently Hoffmann *et al.* introduced several dynamic lexicon features created from Web lists to help train extractors for Wikipedia infobox relations [61]. They shew that these features dramatically improved extractors' performance, especially for sparsely-populated classes.

Other Wikipedia-Related Systems: Völkel *et al.* proposed an extension to be integrated with Wikipedia, which allows the typing of links between articles and the specification of typed data inside the articles in an easy-to-use manner [104]. Though a great step towards semantifying Wikipedia, it still relies on manual labelling by human users. Gabrilovich *et al.* used Wikipedia to enhance text categorization [53], and later proposed a semantic-relatedness metric using Wikipedia-based explicit semantic analysis [54]. Ponzetto *et al.* derived a large scale taxonomy containing subsumption relations based on the category system in Wikipedia [87]. Adafre *et al.* tried to discover missing links in Wikipedia by first computing a cluster of highly similar pages around a target page, then identifying candidate links from those similar pages [5]. Milne *et al.* implemented a new search engine interface called Koru, which harnesses Wikipedia to provide domain-independent, knowledge-based information retrieval [77]. Adler and Alfaro proposed a reputation system for Wikipedia which checks whether users' edits are preserved by subsequent authors [7]. DeRose *et al.* proposed a Cwiki approach to combine both machine and human's contributions to build community portals such as Wikipedia [43]. Dakka and Cucerzan trained a classifier to label Wikipedia pages with standard named entity tags [38].

2.5 Conclusion

This chapter described KYLIN, a relation-specific information extraction system trained using Wikipedia. Since KYLIN uses self-supervised learning, which is bootstrapped on existing user-contributed data, it requires little or no human guidance. We identify some unique

challenges (lack of redundancy) and opportunities (unique identifiers, user-supplied training data, lists, categories, etc.) of this approach. We also identify additional issues resulting from Wikipedia's growth through decentralized authoring (e.g., inconsistency, schema drift, etc.). This high-level analysis should benefit future work on Wikipedia and similar collaborative knowledge repositories. KYLIN uses the Wikipedia heuristic of matching infoboxes with sentences to automatically label training examples to learn relation extractors, and achieves performance which is roughly comparable with that of human editors. In one case, KYLIN does even better.

Although our objective is the automatic extraction of structured data from natural-language text on Wikipedia and eventually the whole Web, our investigation has uncovered some lessons that directly benefit Wikipedia and similar collaborative knowledge repositories. Specifically, Wikipedia could greatly improve consistency if it were augmented with a software robot (perhaps based on KYLIN) which functioned as an automatic fact-checker. When an article was created or edited, this agent could: 1) suggest new entries for an associated infobox; 2) Detect inconsistencies between the text and infobox attributes; 3) note schema inconsistencies in infoboxes and suggest attribute names which have been used previously.

Chapter 3

KOG: WIKIPEDIA INFOBOX ONTOLOGY GENERATION

The vision of a Semantic Web will only be realized when there is a much greater volume of structured data available to power advanced applications. We argue that Wikipedia, one of the world's most popular Websites[1], is a logical source for extraction, since it is both comprehensive and high-quality. Indeed, collaborative editing by myriad users has already resulted in the creation of *infoboxes* for numerous articles. DBpedia [11] has aggregated this infobox data, yielding over 15 million pieces of information.

Furthermore, one may use this infobox data to bootstrap a process for generating additional structured data from Wikipedia. For example, our autonomous KYLIN trained machine-learning algorithms on the infobox data, yielding extractors which can accurately[2] generate infoboxes for articles which don't yet have them. We estimate that this approach can add over 10 million additional facts to those already incorporated into DBpedia. By running the learned extractors on a wider range of Web text and validating with statistical tests (as pioneered in the KnowItAll system [51]), one could gather even more structured data.

In order to effectively exploit extracted data, however, the triples must be organized using a clean and consistent ontology. Unfortunately, while Wikipedia has a category system for articles, the facility is noisy, redundant, incomplete, inconsistent and of very limited value for our purposes. Better taxonomies exist, of course, such as WordNet [1], but these don't have the rich attribute structure found in Wikipedia.

---

[1] Ranked 8th in December 2009 according to comScore World Metrix

[2] KYLIN's precision ranges from mid-70s to high-90s percent, depending on the attribute type and infobox class.

3.1 KOG: Generating the Wikipedia Ontology

This chapter presents the Kylin Ontology Generator (KOG), an autonomous system that builds a rich ontology by combining Wikipedia infoboxes with WordNet using statistical-relational machine learning. Each infobox template is treated as a class, and the slots of the template are considered as attributes. Applying a Markov Logic Networks (MLNs) model [93], KOG uses joint inference to predict subsumption relationships between infobox classes while simultaneously mapping the classes to WordNet nodes. KOG also maps attributes between related classes, allowing property inheritance.

3.1.1 Why an Ontology is Important

We aim to create an ontology which includes the following components:

- **Classes:** types of objects which are semantically distant;
- **Instances:** instances or objects belong to each class;
- **Attributes:** a rich schema listing a comprehensive set of informative attributes for each class. The type information for attributes should also be included;
- **ISA hierarchy:** a well-defined ISA hierarchy over classes;
- **Attribute mappings:** mappings of semantically equivalent attributes between parent/child classes in the ISA hierarchy.

Figure 3.1 shows a fragment of one such ontology. While researchers have manually created ontologies, such as [35], this is laborious and requires continual maintenance. We seek *automatic* ontology construction, which has the potential to better scale as corpora (e.g. Wikipedia in our case) evolve over time. Situating extracted facts in one such ontology has several benefits:

Advanced Query Capability: One of the main advantages of extracting structured data from Wikipedia's raw text is the ability to go beyond keyword queries and ask SQL-like

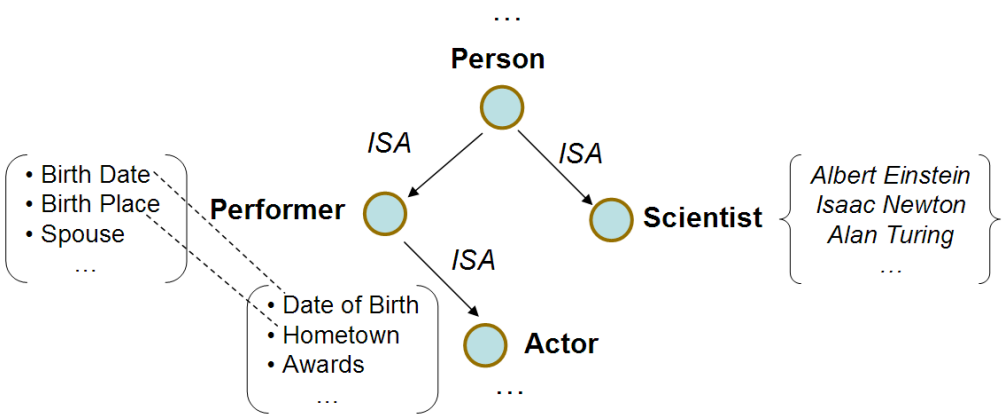


Figure 3.1: A fragment of the ontology created by KOG, which includes an ISA hierarchy over classes, an instance set for each class, an attribute list for each class, and the attribute mappings between parent/child classes.

questions such as "What scientists born before 1920 won the Nobel prize?" An ontology can greatly increase the recall of such queries by supporting transitivity and other types of inference (as we show in section 3.6 later). For example, without recognizing that *particle physicist* is a subclass of *physicist* which is itself a subclass of *scientist*, a Wikipedia question-answering system would fail to return "Arthur Compton" in response to the question above. In many cases the attributes of different Wikipedia infobox classes are mismatched, for example one infobox class might have a "birth place" attribute while another has "cityofbirth" — matching corresponding attributes for subclasses is clearly essential for high recall.

Improving Extractors with Shrinkage: As long as an infobox class has many instances (articles), KYLIN has sufficient training data to learn an accurate extractor. Unfortunately, long-tail distributions mean that most infobox classes *don't* have many instances. When learning an extractor for such a sparsely-populated class, C, one may apply the *shrinkage* technique that uses instances of the parent and children of C, appropriately weighted, as additional training examples. We discuss this in more details in Section 4.1.

Faceted Browsing: When referring to Wikipedia, readers use a mixture of search and browsing. A clear taxonomy and aligned attributes enable faceted browsing, a powerful and popular way to investigate sets of articles [111].

Verifying Extracted Tuples: The type constraints on the arguments of an attribute (relation), *i.e.* selectional preferences, encode the set of admissible argument values. For example, locations are likely to appear as the objects of the "headquarter" attribute and companies or organizations are likely to be the subjects. By checking whether the arguments of extracted tuples have valid types, we can filter those incorrect extractions.

Semiautomatic Generation of New Templates: Today, Wikipedia infobox templates are designed manually with an error-prone "copy and edit" process. By displaying infobox classes in the context of a clean taxonomy, duplication and schema drift could be minimized. Base templates could be automatically suggested by inheriting attributes from the class' parent. Furthermore, by applying the extractors which Kylin learned for the parent class' attributes, one could automatically populate instances of the new infobox with candidate attribute values for human validation.

Infobox Migration: As Wikipedia evolves, authors are constantly reclassifying articles, which entails an error-prone conversion of articles from one infobox class to another. For example, our analysis of five Wikipedia dump "snapshots" between 9/25/06 and 7/16/07 shows an average of 3200 conversions per month; this number will only grow as Wikipedia continues to grow. An editing tool that exploited KOG's automatically-derived schema mappings might greatly speed this process, while reducing errors.

3.2 Kylin Ontology Generator

The KOG system creates a rich ontology[3] by combining evidence from Wikipedia infoboxes and WordNet. As shown in Figure 3.2, KOG is comprised of three modules: the schema cleaner, subsumption detector, and schema mapper.

The *schema cleaner* performs several functions. First, it merges duplicate classes and attributes. Second, it renames uncommon class and attribute names, such as from "ABL" to "Australian Baseball League". Third, it prunes rarely-used classes and attributes. Finally, it infers the type signature of each attribute.

The *subsumption detector* identifies subsumption relationships between infobox classes,

[3]Available at http://ai.cs.washington.edu/www/media/downloadable/media/WikipediaInfoboxOntology-byKOG.zip

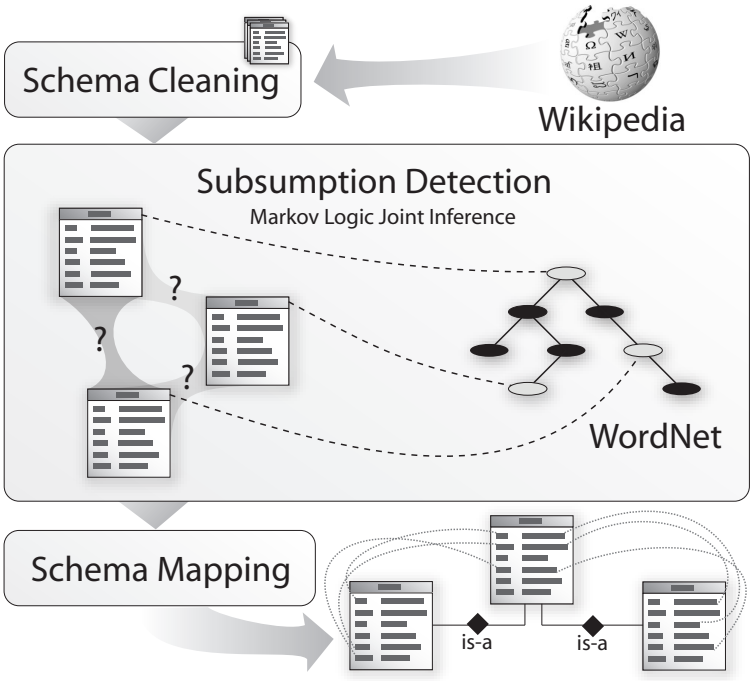


Figure 3.2: Architecture of KOG.

as illustrated in Figure 3.3. We model this as a binary classification problem: given a pair of classes, like "Performer" and "Person", KOG predicts whether the *ISA* relationship holds in between. We apply both the Support Vector Machine (SVM) and Markov Logic Netowrks (MLNs) models using wide range of different features: TF/IDF-style similarity, the WordNet mapping, Wikipedia category tags, Web query statistics, and the edit history of the individual articles.

The *schema mapper* builds attribute mappings between parent/child pairs in the subsumption hierarchy, as illustrated in Figure 3.4. Wikipedia's edit history is essential to this process.

Section 3.6 reports on several alternative designs for these modules. Our experiments show that a joint inference approach, which simultaneously predicts the ISA relationship, while mapping classes to WordNet, achieves the best performance. The next three sections provide details for each module.

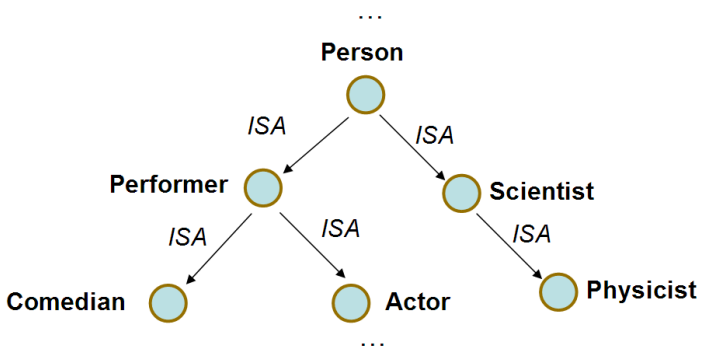


Figure 3.3: The subsumption detector identifies ISA relationships between infobox classes.

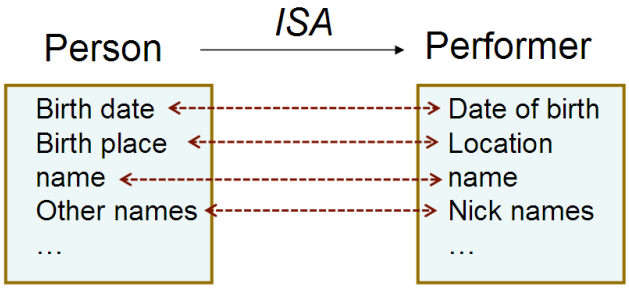


Figure 3.4: The schema mapper builds attribute mappings between parent/child class pair in the subsumption hierarchy.

3.3 Schema Cleaning

As described in Section 2.1.1, the free authoring style in Wikipedia results in a very noisy infobox schemata with duplicate classes and attributes, sparse instantiation, obscure class names, and untyped attributes. Besides filtering rare classes and attributes as done in KYLIN (Section 2.2.1), KOG performs three more tasks when cleaning infobox schemata: identifying duplicate classes and attributes, assigning meaningful class names, and inferring attribute types. Figure 3.5 shows the compact pseudocode of this process.

Schema Cleaning in KOG (Wikipedia Corpus):
IB = {}
1.Recognize Duplicate Schemata:
  For each class C
    If (C's template page is not redirected to another class's template page &&
      C has the largest number of instance articles in classes of the same canonical forms)
      A(C) = {C.a_i}
      For each attribute C.a_i in A(C)
        If C.a_i and C.a_j are semantically equivalent based on edit history
          Delete C.a_i from A(C)
    Add <C, A(C)> to IB
2. Ignore Rare Classes and Attributes:
  For each class C in IB
    If C has few instance articles
      Delete <C, A(C)> from IB
    Else
      For each attribute C.a_i in A(C)
        If C.a_i is used by a small portion of instance articles of C
          Delete C.a_i from A(C)
3. Assign Meaningful Names:
  For each class C in IB
    If C has obscure name
      Rename C using case boundary information, spell checking, aggregated category
        tags and Wikipedia article names.
    For each C.a_i in A(C)
      If C.a_i has obscure name
        Rename C.a_i using similar heuristics as in the previous step.
4. Infer Attribute Types
  For each <C, A(C)> in IB
    For each C.a_i in A(C)
      Infer the type of C.a_i using YAGO and DBpedia mappings to WordNet based on
        the set of attribute values for C.a_i from instance articles of C
Return IB

Figure 3.5: KOG cleans infobox schemata by identifying duplicate classes and attributes, ignoring rare classes and attributes, assigning meaningful names, and inferring attribute types.

3.3.1 Recognizing Duplicate Schemata

One challenge stems from the need to group distinct but nearly-identical schemata. In the case of Wikipedia, users are free to modify infobox templates allowing schema evolution during the course of authoring and causing the problem of class/attribute duplication. For example, four different templates: "U.S. County" (1428), "US County" (574), "Counties" (50), and "County" (19), were used to describe the same class in the 2/06/07 snapshot of Wikipedia. Similarly, multiple tags are used to denote the same semantic attributes (e.g., "Census Yr", "Census Estimate Yr", "Census Est." and "Census Year").

Schema matching has been extensively studied in the data-management community, and the resulting techniques apply directly to the task of duplicate detection [46, 71]. In the case of Wikipedia, however, the task is facilitated by additional features from the collaborative authoring process: redirection pages and the edit history.

Wikipedia uses redirection pages to map synonyms to a single article. For example, "Peking" is redirected to "Beijing". By checking all redirection pages, KOG notes when one infobox template redirects to another.

Next, KOG converts class names to a canonical form: parentheses are replaced with "of" (e.g., "highschool (american)" to "highschool of american"). Underscores, "_", are replaced with a space and digits are discarded (e.g., "musical artist_2" to "musical artist"[4]). Finally, all tokens are converted to lowercase. Classes mapping to the same canonical form are considered to be duplicates.

Conveniently, Wikipedia records a full edit history of changes to the site; KOG exploits this information to locate duplicate attributes within each class. Specifically, we define *transfer frequency*, $f_E(a \rightarrow b)$, as the number of times authors have renamed a to b. For example, if authors have renamed one attribute $C.a_i$ (e.g. "Person.birthPlace") to another $C.a_j$ (e.g. "Person.hometown") more times than a threshold (e.g. 5) then this suggests that they likely denote the same semantics and could be treated as duplicates. Similarly, if two attributes, $C_1.a_i$ and $C_1.a_j$, never have values filled simultaneously, and they have

[4]Sometimes authors add digits to names to indicate a minor difference. Inspection suggests that the variants should be merged when creating a general purpose ontology.

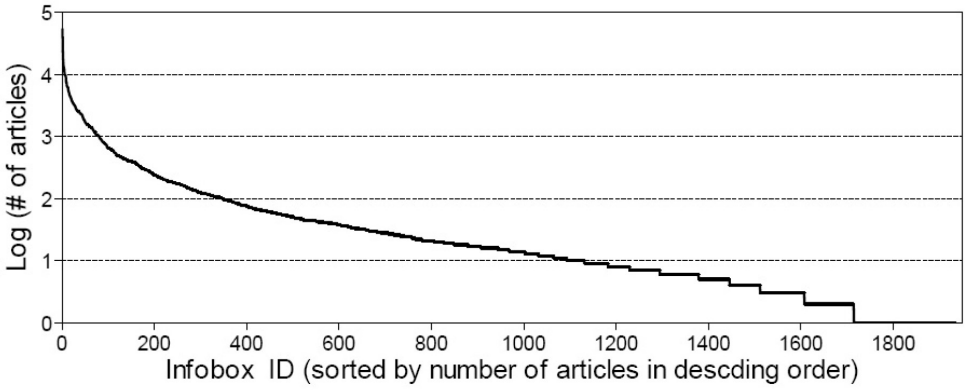


Figure 3.6: The number of article instances per infobox class is similar to a Zipf distribution.

been both transferred to the same attribute of another class $C_2.a_k$, this edit pattern also indicates duplication. KOG combines these edit-history features with the evidence from canonical forms to render its final match.

3.3.2 Ignoring Rare Classes and Attributes

Another consequence of free authoring is schema sparseness — many classes and attributes are used rarely, leading to a long-tailed distribution. For example, Figure 3.6 shows the number of Wikipedia article instances (log scale) per infobox class. Of the 1935 classes, 25% have fewer than 5 instances and 11% have only one. The case is even worse for infobox-class attributes — only 46% of attributes are used by at least 15% of the instances in their class.

We observed that rare infobox classes and attributes often indicate non-representative uses (e.g., the "mayor" attribute for "U.S. County"), or result from noisy data — As a first step, KOG eliminates them from processing. Currently, KOG uses simple statistics for pruning, as done in KYLIN — infobox classes with fewer than 5 articles are ignored. For each class, we keep only those attributes used by more than 15% of instance articles. In the future, we plan more sophisticated methods for dealing with rare items.

3.3.3 Assigning Meaningful Names

In Wikipedia a number of infobox classes have obscure or terse names, such as "ABL," mentioned earlier, and "pref gr," which denotes "prefectures of Greece." Even humans may find it difficult to discern the underlying meaning, yet without an intuitive name a class has limited value.

KOG detects which names may need renaming by identifying those which are missing (even after stemming) from a dictionary, WordNet in our case. If any token from a class name fails to match a WordNet node, KOG passes the name to the four-step procedure described below. If this procedure succeeds at any step, it terminates and returns the recovered full name.

Step 1: Split the name using case boundary information. For example, "horseRacers" would be split into "horse Racers."

Step 2: Use spell checking (i.e., the statistically-based *GoogleSpellSuggestion* function) to find transmutations, e.g. correcting "hungerstriker" to "hunger striker."

Step 3: Collect the category tags of all instance articles within the class, and pick the most frequent k (2 in our case) tags. If the abbreviated form of one tag matches the class name, the tag is treated as the recovered name. Otherwise, the most frequent tag is returned. With the "ABL" class, for example, "Australian Baseball Team" and "Australian Baseball League" are the two most frequent tags, and "Australian Baseball League" would be returned.

Step 4: Query Wikipedia to see if there is an article corresponding to the given class. If it is a redirected page and the title has a good form (as measured heuristically), such as "Video Game" redirected from "cvg", KOG uses the title for the new class name. Otherwise, it uses the definition phrase in the first sentence of the article as the final result. For example, for the "amphoe" class, there is an "Amphoe" article whose first sentence reads "An amphoe is the second level administrative subdivision of Thailand," and so KOG uses *"second level administrative subdivision of Thailand"* as the class name.

These four heuristics may also be used to rename obscurely named *attributes*, such as "yrcom," (year of commission). In addition, KOG uses Wikipedia's edit history to

see if people have manually renamed the attribute in some instances of the class. For example, "stat_pop" can be renamed "population estimation," because users have made some transfers between these two attributes.

3.3.4 Inferring Attribute Types

Even though infobox classes have associated schemata, there is no type system for attribute values. Indeed, since infoboxes are intended solely to provide convenient visual summaries for human readers, there is no *guarantee* that users are consistent with datatypes. Yet inspection shows that most attributes *do* have an implicit type (e.g. "spouse" has type "person"), and if types could be inferred, they would greatly facilitate extraction, fact checking, integration, etc.

KOG infers attribute types from the corresponding set of values, using the following five-step procedure:

Step 1: Let c be an infobox class with attribute a. For example, a might be the "spouse" attribute of the "person" class. KOG generates the set of possible values of a, and finds the corresponding Wikipedia objects, $V_{c,a}$; note not every value will have corresponding Wikipedia article.

Step 2: Create a partial function $\omega : V_{c,a} \rightarrow N_w$ from value objects to WordNet nodes by combining two preexisting partial mappings. The first source, "DBpediaMap," is DBpedia's [11] manually created mapping from 287,676 articles to corresponding WordNet nodes. If DBpediaMap does not have an image for $v \in V_{c,a}$, then KOG uses "YagoMap," an automatically-created mapping, which links a greater number, 1,227,023, of Wikipedia articles to WordNet nodes [103].

Step 3: Consider the set of WordNet nodes $\{n$: there exist at least t distinct $v \in V_{c,a}$ such that $\omega(v) = n\}$ for some threshold, t (we use $t = 10$). If there are at least 2 nodes in this set, KOG considers the two which are mapped by the most values in $V_{c,a}$ and finds their relationship in WordNet. If the relationship is *alternative, sibling, or parent/child*, KOG returns their least common parent synset as the final type for the given attribute. For example, if the two most frequent nodes are "physicist" and "mathematician", then KOG

would choose type "scientist," because it is the direct parent of those two siblings. If no relationship is found, KOG sets the type equal to the synset of the most frequent node.

Step 4: If no WordNet node is mapped by at least t values in $V_{c,a}$, KOG creates a larger set of values, V, by adding the values of a similar class, c' which also has attribute a. For example, Wikipedia entities from "Person.Spouse" and "Actor.Spouse" would be put together to compute the accumulated frequency. The most frequent WordNet node would be returned as the type of the target attribute.

Step 5: If Step 4 also fails, KOG analyzes the edit history to find the most related attribute, which has the highest number of transfers with the target attribute. The type of this most-related attribute is then returned as the type of a.

KOG can also generate a type signature for a complete infobox class. Indeed, this is easy after the class has been mapped to a WordNet node, which is described in the next section.

3.4 Subsumption Detection

Detecting subsumption relations, *i.e.* that one class *ISA* subset of another, is the most important challenge for KOG. Figure 3.7 shows the compact pseudocode of KOG's subsumption detection module. We model this task as a binary classification problem and use machine learning to solve it. Thus, the two key questions are: 1) which features to use, and 2) which machine learning algorithm to apply. In fact, we implemented two very different learning frameworks: SVMs and a joint inference approach based on Markov logic. The next subsection defines the features: a mixture of similarity metrics and Boolean functions. Section 3.6 shows that our joint inference approach performs substantially better.

3.4.1 Features for Classification

KOG uses six kinds of features, some metric and some Boolean.

Similarity Measures: Class similarity is an indication of subsumption, though not a sufficient condition. KOG uses four different similarity metrics.

Attribute Set Similarity: KOG models a class as the set of its attributes, compresses each set into a bag of words, and computes the TF/IDF similarity score between the bags.

Subsumption Detection in KOG (cleaned infobox schemata IB, WordNet WN):
SD = {}
1.Create Feature Set:
  For each pair of classes <C_i, C_j> in IB
    Create six kinds of features (metric or Boolean)
    Encode features to MLNs formulas
2. Apply MLNs Model to Jointly Infer:
  Whether "C_i ISA C_j" for each pair of <C_i, C_j>; add cases of "(C_i ISA C_j)=true" to SD
  Whether "C_i maps to N_i in WN"; add cases of "(C_i mapTo N_i)=true" to SD
3. Construct ISA Tree
  Topological Sorting over {(C_i ISA C_j)=true}
  For each "(C_i ISA C_j)=true" in SD
    If C_j is not the closest ancestor of C_i in the sorted sequence
      Delete "(C_i ISA C_j)=true"
Return SD

Figure 3.7: KOG simultaneously predicts the ISA relationship between infobox classes and maps classes to WordNet nodes via joint inference using Markov Logic Networks.

First Sentence Set Similarity: For each class, KOG creates a bag of words by taking the first sentences of each instance of the class. Once again, the TF/IDF score between the bags defines the class similarity.

Category Set Similarity: The bags are created from the category tags attached to the class instances.

Undirected Transfer Frequency: This similarity score is computed from Wikipedia's edit history. If c and c' denote two classes, define their *undirected transfer frequency* as $f_E(c \leftrightarrow c') = f_E(c \rightarrow c') + f_E(c' \rightarrow c)$. We normalize this frequency to $[0, 1.0]$.

Class-Name String Inclusion: Inspired by [105], we say that the feature *isaFT(c,d,Contain)* holds iff: 1) the name of d is a substring of c's name, and 2) the two names have the same head (as determined by the Stanford Parser [3]). For example, the feature holds for $c =$ "English public school" and $d =$ "public school," since both have "school" as head.

Category Tags: Many infobox classes have their own Wikipedia pages, and sometimes a special type of category, "XXX infobox templates," is used to tag those pages. We say that the feature *isaFT(c,d,HasCategory)* holds if class c has a special category tag called

"⟨name of d⟩ infobox templates."

For example, the page for the "volleyball player" class has a category tag called "athlete infobox templates," and there exists another class named "athlete," so *isaFT("volleyball player", "atheletes", HasCategory)*. This feature is strongly linked to subsumption (e.g., "volleyball player" *ISA* "athlete," but nothing is guaranteed. For example, both "athlete" and "Olympic" classes have the category tag "Sports infobox templates", but neither of them *ISA* sports.

Edit History: The edit patterns from Wikipedia's evolution contain useful information — intuitively, when changing the type of an instance, an author is more likely to specialize than generalize. We define the *degree* of class c as the number of classes transferring with c. Given a pair of classes c and d, KOG checks: 1) whether $f_E(c \leftrightarrow d)$ is high enough (e.g, bigger than 5 in our case); 2) Whether the degree of d is much bigger than that of c (e.g. more than twice as big). If both conditions are true, we say the feature *isaFT(c,d,EditH)* holds — weak evidence for *"c ISA d"*.

Hearst Patterns: Following [51, 58], KOG queries Google to collect type information about class names using patterns such as *"NP0, like NP1"* and *"NP0 such as NP1"*, which often match phrases such as "...scientists such as phsyicists, chemists, and geologists." We say *isaFT(c,d,HPattern)* holds if the Google hit number for HPattern(c,d) is big enough(e.g. 200 in our case) while very small for HPattern(d,c)(e.g. less than 10 in our case).

WordNet Mapping: By computing a mapping from infobox classes to WordNet concept nodes, KOG gains useful features for predicting subsumption. For example, if both c and d have perfectly corresponding nodes in WordNet and one WordNet node subsumes the other (say *isaFT(c,d,isaWN)*), then this is likely to be highly predictive for a learner. Since computing the mapping to WordNet is involved, we describe it in the next subsection.

3.4.2 Computing the WordNet Mapping

In this section we explain how KOG generates two mappings between infobox classes and WordNet nodes: $\omega(c)$ returns a node whose name closely matches the name of c, while $\varphi(c)$ denotes the node which most frequently characterizes the *instances* of c according to

Yago [103]. Based on these two mappings, KOG seeks the closest semantic match $\varpi(c)$ for each class c in WordNet (e.g., "scientist" class should map to the "scientist" node instead of the "person" node), and outputs one of the seven mapping types characterizing different degrees of match; in descending order of strength we have: *LongName*, *LongHead*, *ShortNameAfterYago*, *ShortHeadAfterYago*, *HeadYago*, *ShortName*, and *ShortHead*. The following steps are tried in order until one succeeds.

Step 1: If class c's name (after cleaning) has more than one token, and has an exact match in WordNet, $\omega(c)$, then $\omega(c)$ is output as the closest semantic match $\varpi(c)$ with mapping type *LongName*. This kind of mapping is very reliable — a random sample of 50 cases showed perfect precision.

Step 2: KOG uses the Stanford Parser to locate the head of c's name and returns the WordNet node which matches the longest substring of that head, $\omega(c)$. For example, "beach volleyball player," is matched to "volleyball player" in WordNet, instead of "player." If the matched head has more than one token, then $\omega(c)$ is returned with type *LongHead*; a sample shows that it is also very reliable.

Step 3: If neither of the previous techniques work, KOG looks for a consensus mapping amongst articles which instantiate the class, much as it did when determining an attribute's type in Section 3.3.4. However, instead of using both the DBpediaMap and YagoMap to define the mapping, as done previously, KOG just uses YagoMap, saving the higher-quality, manually-generated DBpediaMap to use as training data for the learner. Let I_c denote the instances of infobox class c; for all $o \in I_c$ let $\varphi(o)$ be the WordNet node defined by Yago. Let $\varphi(c)$ be the most common node in $\varphi(I_c)$. If c's head is a single token, and has a matched node $\omega(c)$ in WordNet, KOG checks the relationship between $\omega(c)$ and $\varphi(c)$ in WordNet. If $\omega(c)$ is a child or alternative of $\varphi(c)$ and the head is the class name itself (i.e., c's name is a single token), KOG returns $\omega(c)$ with type *ShortNameAfterYago*; otherwise, KOG returns $\omega(c)$ with type *ShortHeadAfterYago*. If no relationship is found between $\omega(c)$ and $\varphi(c)$, KOG returns $\varphi(c)$ with type *HeadYago*. If no $\varphi(c)$ is found, KOG returns $\omega(c)$ with type of either *ShortName* or *ShortHead*, depending on whether c is single token. As in Yago [103], we select the most frequent sense of the mapped node in WordNet, which turns out to work well in most cases.

To finally determine whether $\varpi(c)$ is returning a good mapping, KOG encodes its mapping type as a Boolean feature: *mapType(c,t)*, where t denotes one of the seven types (e.g., *LongName*). A support vector machine (SVM) is learned using DBpediaMap as a training set (We used the LIBSVM implementation [4]). In this way, the SVM learns relative confidences for each mapping type and outputs a score for the WordNet mappings. This score can be used to easily control the precision / recall tradeoff. Furthermore, the score could also identify potentially incorrect mappings for further verification (whether manual or automatic).

Now, when given two classes c and d, KOG can check whether $\varpi(c)$ subsumes $\varpi(d)$ in WordNet. If so, KOG constructs the feature, *isaFT(c,d,isaWN)*, which is likely to be highly predictive for the subsumption classifier described next.

We close by noting that, in addition to being a useful feature for the subsumption classifier, the WordNet mapping has other important benefits. For example, each node in WordNet has an associated set of synonyms which can be used for query expansion during query processing over the infobox knowledge base. For example, consider a query about ballplayers born in a given year. Even though there is no "ballplayer" class in Wikipedia, WordNet knows that "ballplayer" and "baseball player" are synonyms and so a query processing system can operate on records of the "baseball player" class. Additionally, associating the attributes from a Wikipedia schema (as well as a long list of class instances) with a WordNet node may also provides substantial benefit to WordNet users as well.

3.4.3 Max-Margin Classification

One might think that there would be no need to learn a second classifier for subsumption, once KOG has learned the mapping from infobox classes to WordNet, but in practice the WordNet mapping is not 100% correct, so the other features improve both precision and recall. But even if the first SVM classifier could learn a correct mapping to WordNet, it would be insufficient. For example, "television actor" and "actor" are both correctly mapped to the WordNet node "person," but this mapping is incapable of predicting that "actor" subsumes "television actor." Instead, KOG treats the mapping as just another

feature and learns the subsumption relation using all available information.

KOG uses the "DBpediaMap" to constructs the training dataset (details in section 3.6.2) to train a SVM classifier for subsumption. By automatically weighing the relative importance of all features, KOG finds an optimal hyperplane for classifying subsumption. With a confidence threshold of 0.5, the SVM achieves an average precision of 97.2% at a recall of 88.6%, which is quite good.

3.4.4 Classification via Joint Inference

While the SVM's performance is quite good, there is still space to improve. First, the SVM classfier predicts *ISA* between each pair of classes sequentially and separately. This local search ignores evidence which is potentially crucial. For example, if *"c ISA d"* and *"d ISA e"*, it is likely that *"c ISA e"*, even if no strong features observed for the pair of c and e.

Secondly, the SVM classifiers separate the WordNet mapping and *ISA* classification as input and output, so that the crosstalk between these two parts is blocked. In reality, however, these two problems are strongly intermixed and evidence from either side can help to resolve the uncertainty of the other side. For example, given that class c and d have correct mappings to WordNet and *"$\varpi(c)$ ISA $\varpi(d)$"*, it is likely that *"c ISA d"*; on the other hand, if *"c ISA d"* but the retrieved mappings $\varpi(c)$ and $\varpi(d)$ have no *ISA* relationship in WordNet, then it is clear that something is wrong — but the SVM won't recognize the problem.

In contrast, a relational-learning model capable of joint inference *can* exploit this global information. To see if this would lead to greater performance, we applied the Markov Logic Networks(MLN) model. By addressing *ISA* classification and WordNet mapping jointly, our MLN model achieves substantially better performance for both tasks. Section 3.6 provides detailed experiments, but we note that with a confidence threshold of 0.5, our MLN model eliminated 43% of the residual error while simultaneously increasing recall from 88.6% to 92.5%.

Before describing our MLN classifier in more detail, we provide background on Markov Logic Networks.

Markov Logic Networks

A first-order knowledge base can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea of MLNs is to soften these constraints: when a world violates one formula in the KB it is deemed less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

(From Richardson & Domingos [93]) A *Markov Logic Network* L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:

1. $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.

2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

Thus, there is an edge between two nodes of $M_{L,C}$ if and only if the corresponding ground predicates appear together in at least one grounding of one formula in L. An MLN can be viewed as a *template* for constructing Markov networks. The probability distribution over possible worlds x specified by the ground Markov network $M_{L,C}$ is given by

$$P(X = x) = \frac{1}{Z} \prod_i \phi_i(x_{\{i\}}) \tag{3.1}$$

where Z is the normalization factor, $\phi_i(x_{\{i\}})$ is the potential function defined on the ith clique which is related to a grounding of formula F_i, and $x_{\{}i\}$ is the discrete variable vector in the clique. Usually, it is represented as follows,

$$\phi_i(x_{\{i\}}) = \begin{cases} e^{w_i} & F_i(x_{\{i\}}) = True \\ 1 & F_i(x_{\{i\}}) = False \end{cases} \tag{3.2}$$

In this way, we can represent the probability as follows,

$$P(X = x) = \frac{1}{Z} \exp\left\{\sum_i w_i n_i(x)\right\} \tag{3.3}$$

where $n_i(x)$ is the number of true groundings of F_i in x.

Using MLNs to Classify Subsumption

KOG uses the open source Alchemy system [65] to implement its MLN classifier. As described in Section 3.4.1, two predicates are used to represent features: *mapType(c,t)* and *isaFT(*c_1*,*c_2*,f)*. Two query predicates *isa(*c_1, c_2*)* and *map(c)* are used to express the uncertainty of *ISA* classification and WordNet mapping, respectively. After learning, Alchemy computes the probabilities of these predicates.

We use three kinds of logical formulas to guide KOG's learning. The first represents the loose connection between WordNet mappings and the corresponding types. For example,

$$mapType(c, LongName) \Leftrightarrow map(c)$$

which means "Class c has a long class name and exactly matches a node in WordNet if and only if this mapping is correct."[5] Remember that Alchemy will learn the best probabilistic weight for this and the other rules. By using a metavariable, "$+t$," instead of the constant *LongName*, we direct Alchemy to learn weights for all possible indications: $mapType(c, +t) \Leftrightarrow map(c)$.

The second class of formulas encode the intuition that 1) *ISA* is transitive, and 2) features such as *isaFT(*c_1*,*c_2*,Contain)* are likely correlated with subsumption:

$$isa(c_1, c_2) \wedge isa(c_2, c_3) \Rightarrow isa(c_1, c_3)$$

$$isaFT(c_1, c_2, +f) \Leftrightarrow isa(c_1, c_2)$$

The final formulas encode the connection between the WordNet mapping and *ISA* classification:

$$isaFT(c_1, c_2, isaWN) \wedge map(c_1) \wedge map(c_2) \Rightarrow isa(c_1, c_2)$$

[5] In fact, Alchemy converts the bidirectional implication into two separate clauses, one for each direction; this allows it to learn different weights for each direction.

which means "if c_1 and c_2 both have correct WordNet mappings and the mapped nodes are *ISA* in WordNet, then c_1 *ISA* c_2."

Two other formulas complete the intuition:

$$isaFT(c_1, c_2, isaWN) \wedge isa(c_1, c_2) \Rightarrow map(c_1) \wedge map(c_2)$$

$$map(c_1) \wedge map(c_2) \wedge isa(c_1, c_2) \Rightarrow isaFT(c_1, c_2, isaWN)$$

Discriminative learning is used to determine the weights of formulas [99], and MC-SAT is used for inference [88]. Experimental results show that this joint inference approach improves the precision of both *ISA* classification and WordNet mapping.

3.4.5 ISA Tree Construction

The SVM and MLNs classifiers perform subsumption detection for all pairs of infobox classes. A class might be predicted as a subset of both its parent and grandparent classes. For example, "actor" is classified as subsuming both "performer" and "person". We apply a topological sorting over the classification results to build a tree structured taxonomy, where each class is only attached to its closest parent. Specifically, we build a directed graph $< V, E >$, where each infobox class is a vertex in V, and E is the set of predicted subsumption. We perform a topological sorting over the graph to get a liner ordering of its nodes in which each node comes before all nodes to which it has outbound edges. In the final taxonomy, a class node is only attached to its closest parent based on the ordering. For example, "actor" is attached to "performer", which itself is attached to "person".

3.5 Schema Mapping

To support property heritance, KOG identifies corresponding attributes between parent and child classes in the ISA hierarchy. Figure 3.8 shows the compact pseudocode for this process. Schema mapping is a well-studied database problem, which seeks to identify corresponding attributes among different relational schemata [46, 71]. With KOG, we take a simple approach which exploits the structure of Wikipedia, relying on the edit history and string similarity comparison to find attribute mappings.

Schema Mapping in KOG (parent class C_1, child class C_2):
SM = {}
For each pair of <$C_1.a_i$, $C_2.a_j$>
    If $C_1.a_i$ and $C_2.a_j$ are semantically equivalent based on edit history
        Add <$C_1.a_i \sim C_2.a_j$> to SM
    Else if the names of $C_1.a_i$ and $C_2.a_j$ are similar enough
        Add <$C_1.a_i \sim C_2.a_j$> to SM
Return SM

Figure 3.8: KOG identifies corresponding attributes between parent and child classes based on Wikipedia's edit history and string similarity comparison.

Let c and d denote two classes, typically a parent/child pair from the subsumption lattice constructed in the previous section. KOG considers different pairs of attributes, looking for a match by checking the following conditions in turn:

Step 1: If the *undirected transfer frequency* between two attributes $c.a$ and $d.b$ is high enough ($f_E(c.a \leftrightarrow d.b) \geq 5$ in our case), KOG matches them.

Step 2: If data is sparse, KOG considers attribute names independent of class, looking at the edit history of all classes with attributes named a and b. For example, it treats "actor.spouse" and "person.spouse" both as "spouse," and "person.wife" and "musician artist.wife" both as "wife," and computes the sum of $f_E(a \leftrightarrow b)$ between all possible pairs of attributes (a, b). If an attribute $c.a$ wasn't mapped in Step 1 and $f_E(a \leftrightarrow b)$ is over threshold in this aggregate fashion, then KOG maps $c.a$ to $d.b$.

Step 3: If the previous steps fail, KOG uses the lexical, string comparison method, like that of Section 3.3.1.

Once mapping is complete, KOG iterates over all attributes, collecting every corresponding attribute into a bag of alternative names. For example, the "birth place" attribute of "person" is given the following alternative names: birthplace, place of birth, place birth, location, origin, cityofbirth, born." This naming information is helpful for query expansion and for other tasks (e.g., query suggestion, information integration, etc.)

Since KOG has already estimated a type signature for each attribute, it uses this to

double-check whether the attribute mapping is consistent. Those which fail to match are tagged for subsequent verification and correction, which could be manual or automatic. In the future, we intend to add the attribute mapping phase, with type consistency, into our joint inference approach.

3.6 Experiments

To investigate KOG's performance, we downloaded the English version of Wikipedia for five dates between 09/25/2006 and 07/16/2007. We evaluate ontology refinement on the 07/16/2007 snapshot; previous versions are used to compute edit-history information. There are many measurements for taxonomy creation. We choose the most general criteria of precision and recall.

3.6.1 Schema Cleaning

This section addresses three questions:

- How does KOG recognize duplicate schemata?
- How does KOG assign meaningful names?
- What is the precision for attribute type inference?

Recognizing Duplicate Schemata

Our data set contained 1934 infobox templates. By following redirected pages, KOG found 205 duplicates; checking canonical forms identified another 57. A manual check yielded an estimate of 100% precision. To estimate recall, we randomly selected 50 classes and found 9 duplicates by manual checking. KOG successfully identified 7 of them, which leads to an estimated recall of 78%. Since KOG also prunes classes containing less than 5 instance articles, 1269 infobox classes are selected.

For attributes, KOG found 5365 duplicates — about 4 per class. We randomly selected 10 classes and manually identified 25 true duplications. On this set KOG predicted 23 duplicates of which 20 of them were correct. This leads to an estimated precision of 87%,

Heuristic	Precision(%)	Recall(%)	F-Measure(%)
CaseCheck	97.0	10.1	18.2
GoogleSpell	91.7	6.9	12.9
Category	86.7	61.3	71.8
WikiQuery	90.0	5.7	10.7
All	88.4	84.0	86.1

Table 3.1: Performance of assigning meaning full class names.

and estimated recall of 79%. Since KOG ignores attributes which are used by less than 15% instance articles, 18406 attributes (out of 40161) survived cleaning.

Assigning Meaningful Names

By referring to WordNet, KOG selected 318 out of 1269 infoboxes for name recovery. KOG found names for 302 of these candidates and manual checking rated 267 of them to be correct. This is quite encouraging, because many class names are extremely hard to interpret — even for human beings. For example, KOG correctly renamed "wfys" to be "youth festivals" and renamed "nycs" to "New York City Subway." Table 3.1 shows the detailed contribution of each heuristic, where "All" means the combination of every heuristic, as described in section 3.3.1.

For attributes, we randomly selected 50 classes which contain a total of 654 attributes. By referring to WordNet, KOG identified 153 candidates, and it reassigned names to 122 of them; 102 of the new names were correct. This leads to an estimated precision of 84% and recall of 67%. We note that KOG didn't perform as well here as it did when renaming class names. One explanation may be that less attention is paid by humans to attribute names, and this provides a weaker signal for KOG to exploit.

Inferring Attribute Types

In order to check the performance of type inference, we randomly picked 20 infobox classes, which had a total of 329 attributes. KOG predicted a type for 282 of these and 186 predictions were correct. This leads to an estimated precision of 66% with a recall of

57%. These results are acceptable given the problem difficulty and lack of labeled training data, but we anticipate that by incorporating the techniques introduced by the REALM model [48], KOG could do substantially better.

3.6.2 ISA Classification

We now focus on three additional questions:

- What are the precision and recall of subsumption detection?
- How does KOG identify incorrect WordNet mappings?
- What is the benefit (if any) of joint inference?

First, however, we describe how KOG automatically derive a training dataset based on open Web resources.

Training dataset construction

Recall that "DBpediaMap" contains manually-created mapping from 287,676 articles to their corresponding WordNet nodes; the articles come from 266 of the infobox classes. KOG uses this data to construct the pseudo-training dataset for subsumption detection[6]. We call it "pseudo" because it is constructed indirectly, as follows. For each class covered by "DBpediaMap", we first select the most frequent aggregated label over its instance articles. Then we decide whether this is a *NameMap* or *HeadMap*: if the label exactly matches the class name or one of its alternative terms in WordNet, we call it a *NameMap*; otherwise call it a *HeadMap*. Besides "DBpediaMap", two other mappings types are also added to the pseudo dataset due to their high precision: one is *LongName* and another *LongHead*. In this way, we get a dataset of 406 classes with pseudo-labelled WordNet mappings. Then KOG produces positive and negative *ISA* pairs by following the hyponym tree in WordNet:

- Suppose both class c and d have *NameMap* $\varpi(c)$ and $\varpi(d)$. If $\varpi(c)$ *ISA* $\varpi(d)$, KOG labels *"c ISA d"*. Otherwise, *"c NOT ISA d"*;

[6]Other datasets, like the automatically compiled "YagoMap" or "Category Ontology" from [87], can also serve for this purpose.

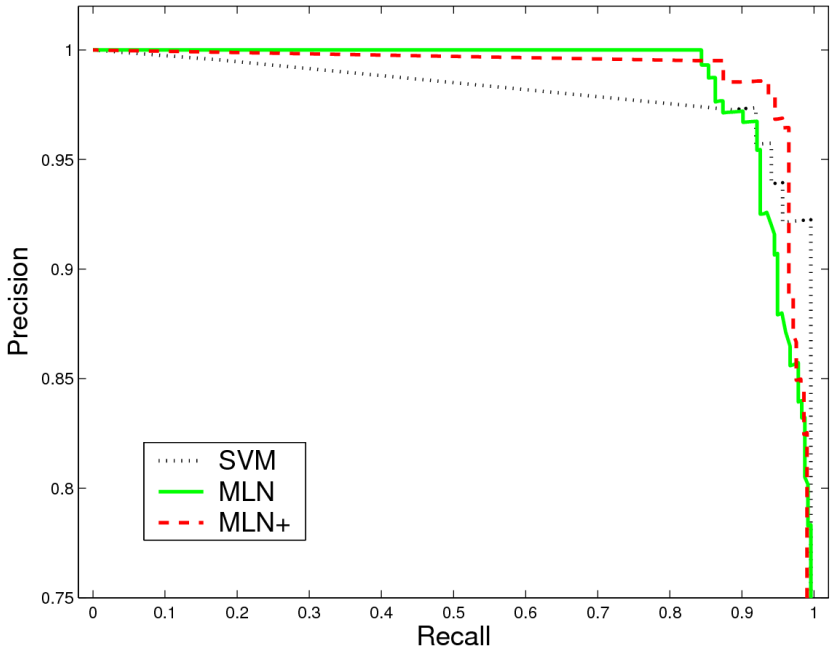


Figure 3.9: Subsumption detection via joint inference using MLNs is superior than applying a SVM model.

- Suppose class c has *HeadMap* $\varpi(c)$ and class d has *NameMap* $\varpi(d)$. If $\varpi(c)$ *ISA* $\varpi(d)$*"*, or $\varpi(c)$ is an alternative term of $\varpi(d)$, we label *"c ISA d"*.

In this way, KOG gets 205 positive and 358 negative ISA pairs for training[7].

Results

To better understand the source of performance at subsumption classification, we also implemented a simplified MLN classifier; it uses exactly the same features as the SVM classifier (without the formulas for crosstalk between WordNet mapping and *ISA* classification). For clarity, we call the simplified model "MLN," and the fully-functional one "MLN+." We test each model's performance with 5-fold cross validation on the pseudo-labelled dataset. Each run is performed on a Linux platform with a 2.4GHz CPU and 4G memory. In average, it takes about 0.05 minutes to train the SVM classifier and 0.2 minutes to test; for the

[7]This benchmark dataset is available at: http://ai.cs.washington.edu/www/media/downloadable/media/ISA-PseudoManual-KOG.zip

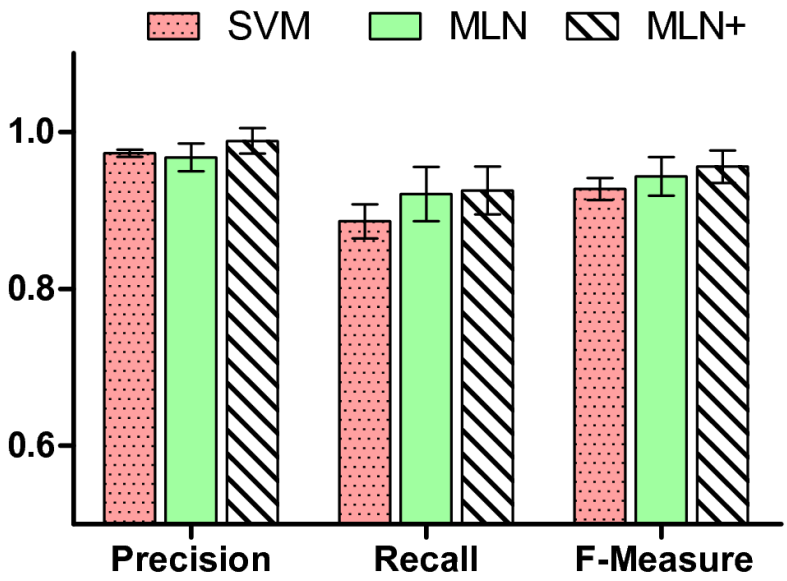


Figure 3.10: ISA classification with confidence threshold set as 0.5.

"MLN+" model, it takes about 6.1 minutes to train (with about 45K grounded predicates and 91K grounded clauses) and 0.6 minutes to test.

Figure 3.9 shows the precision/recall curves for subsumption classification. All three models perform well. We suspect most people are willing to trade away recall for higher precision. In this sense, both MLN models perform better than the SVM classifier, and MLN+ is the best by further extending the recall. To have a close look, we set the confidence threshold at 0.5, and compared three models' performance in Figure 3.10. The SVM classifier achieves an excellent precision of 97.2% and recall of 88.6%. The MLN model drops precision to 96.8% but has better recall at 92.1%. And MLN+ wins on both counts, extending precision to 98.8% (eliminating residual error by 43%) and recall to 92.5%. Since the only difference between the two MLNs are the formulas inducing joint inference, it is clear that this is responsible.

As we mentioned before, the WordNet mapping is useful in its own right. To check how joint inference affects this task, we again implemented a simplified MLN and compared the performance of three models. Figure 3.11 shows that both MLN models achieve a big

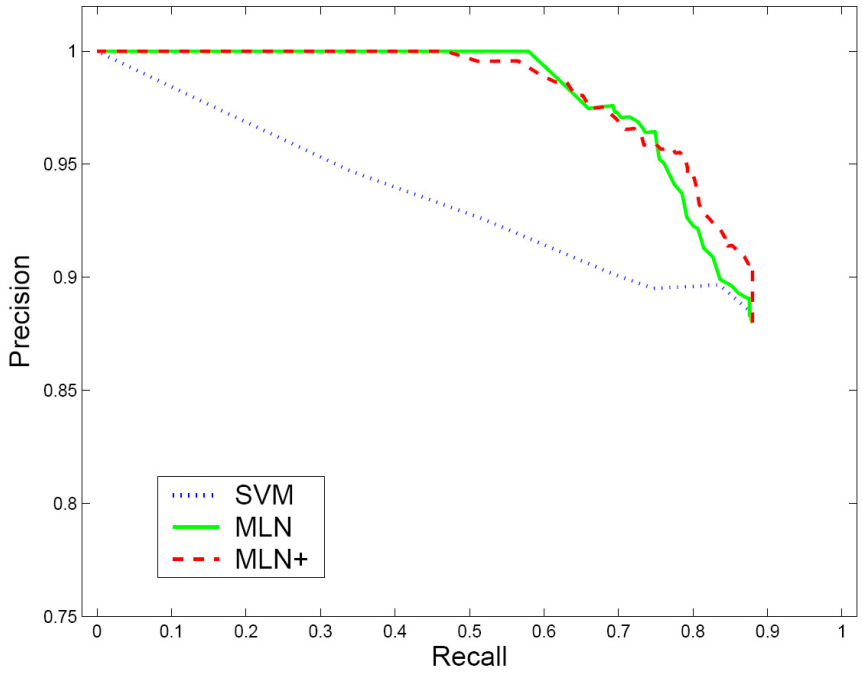


Figure 3.11: WordNet mapping via joint inference using MLNs achieves better performance than applying a SVM model.

improvement over the SVM classifier.The MLN+ model has overall better performance than MLN, especially at high recall. This improvement stems from MLN+'s ability to identifying incorrect WordNet mappings, as shown in Figure 3.12. This ability may translate into an effective *mixed-initiative interface*, since the MLN+ model will be able to drive active learning, asking humans to correct examples which it knows are incorrect.

3.6.3 Schema Mapping

This section evaluates the precision and recall of KOG's schema mapping capability. In particular, we are interested in the ability to accurately map corresponding attributes between parent and child schemata. To perform the evaluation, we took a random sample of 10 *ISA* class pairs from the constructed subsumption lattice[8]. Manual checking revealed a total of

[8]Available at: http://ai.cs.washington.edu/www/media/downloadable/media/schemaMappingEvaluation_gold-standard.zip

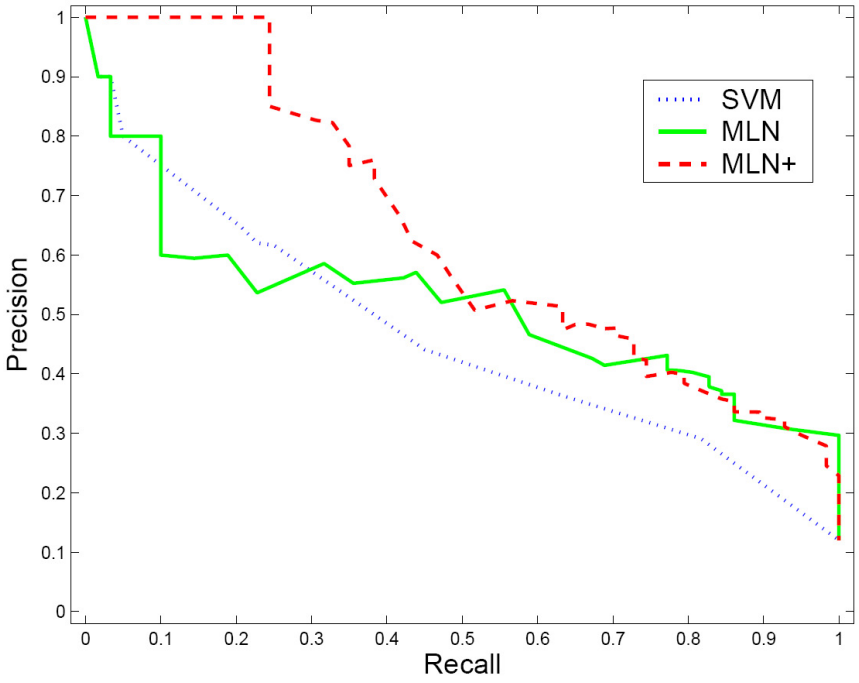


Figure 3.12: MLNs are more effective than SVM for detecting incorrect WordNet mappings.

91 true mappings. KOG made 84 predictions and 79 of them were correct. This leads to an estimated precision of 94% and recall of 87%. There are two main causes for incorrect mappings: first, some ambiguous attributes are invented for flexible visualization purpose. For example, "free" and "free_label" are widely used by users to define attributes. This ambiguity misleads KOG to link several attributes to these free attributes. Secondly, the string-similarity heuristic also produces errors occasionally. For example, "road.direction_a" is mapped to "route.direction_b" since only one character is different. In the future we hope to incorporate recently-developed schema-mapping techniques from the database community in order to boost precision and recall.

3.6.4 Sample Application

The main motivation for KOG was the hope that the resulting ontology would support advanced queries over data extracted from Wikipedia and the Web. As preliminary confirmation of this, we did a case study to check its support for query over Wikipedia infoboxes.

It turns out KOG helps to extend the recall in many cases. For example, given a query like:

- *"Which performing artists were born in Chicago?"*

Without the refined ontology, one would likely return zero results, because there is no "performing artist" infobox in the current Wikipedia. However, with KOG we know "performer" is an alternative of "performing artist" and its "location" attribute has "born" as an alias. As a result, the answer "Michael Ian Black" would be successfully retrieved from an infobox. Furthermore, by following the ISA tree, we know "actor" and "comedian" are children of "performer", and their attributes "birthplace", "birth place", "cityofbirth", "place of birth", "origin" are duplicates, all mapping to the "location" attribute of "performer." This expansion allows the return of 162 additional answers from "actor" and one additional answer from the "comedian" class.

3.7 Related Work

Ontology Construction Based on Wikipedia: Suchanek *et al.* built the Yago system by unifying WordNet and Wikipedia, where leaf category tags are mapped to WordNet nodes with rule-based and heuristic methods [103]. Strube *et al.* derived a large scale taxonomy based on the Wikipedia category system by applying several heuristics to identify the ISA relationships among category tags [87]. In contrast with this work, we focus on combining Wikipedia infoboxes with WordNet, and trained a sophisticated MLN model to jointly infer the WordNet mapping and ISA classification. In some sense, their work and ours are complementary to each other: they achieve greater coverage with category tags, and we provide detailed schemata together with attribute mappings.

Herbelot *et al.* extracted ontological relationships from Wikipedia's biological texts, based on a semantic representation derived from the RMRS parser [60]. In contrast, KOG constructs a broad, general-purpose ontology. Hepp *et al.* propose to use standard Wiki technology as an ontology-engineering workbench and show an application of treating Wikipedia entries as ontology elements [59]. We are also motivated by the special structures (e.g., infoboxes) in Wikipedia, and try to address more advanced problems, such as subsumption extraction and schema mapping.

Learning Relations from Heterogenous Evidence: Cimiano *et al.* trained an SVM classifier to predict taxonomic relations between terms by considering features from multiple and heterogeneous sources of evidence [34]. For KOG, we also used SVM classifiers to handle diverse features from heterogenous evidences. Furthermore, we also applied an MLN model, showing the benefit of joint inference: by using a single MLN classifier, KOG creates the WordNet mapping and ISA classification simultaneously — getting better performance on both tasks.

Snow *et al.*'s work [100] is closer to ours in the sense of handling uncertainty from semantic relationships and WordNet mappings all together over heterogenous evidence. However there are several important differences. First, they use local search to incrementally add one new relation in each step, greedily maximizing the one-step increase in likelihood. This hill-climing model risks slipping into a local maximum, with no ability to jump to a globally better solution. In contrast, we use a MLN model to jointly infer the value of all relations, more likely finding the optimal solution. Second, Snow et al. assume that each item of evidence is independent of all others given the taxonomy, and depends on the taxonomy only by way of the corresponding relation. In contrast, our MLN model doesn't make any independence assumption during inference.

Schema Matching: Several of the problems addressed by KOG may be seen as instances of the schema-matching problem: recognizing duplicate schemata, finding duplicate attributes, and matching the attributes of one schema with those of a subsuming class. Many researchers have investigated this general problem, especially those in the database and IR communities. For example, Doan *et al.* developed a solution combining several types of machine learning [46]. Madhavan *et al.* proposed a corpus-based matching approach which leverages a large set of schemata and mappings in a particular domain to improve robustness of matching algorithms [71]. He and Chang proposed a statistical schema mapping framework across Web query interfaces by integrating large numbers of data sources on the Internet [57]. Bilke and Naumann exploit the existence of duplicates within data sets to perform automatic attribute mappings [20]. We would like to incorporate these approaches into KOG, but to date have implemented a simpler, heuristic approach which

exploits Wikipedia-specific structure to yield acceptable performance.

General Ontology Construction: The most widely used method for automatic ontology extraction is by lexico-syntactic pattern analysis. This is first proposed by Marti Hearst to acquire hyponyms from large text corpora [58], and later followed by many successful systems, such as KnowItAll [51] and PANKOW [31, 32]. Cafarella *et al.* propsed the TGen system to discover schemas from the unstructured assertions harvested from the Web [28]. Another general way to learn ontology is clustering concept hierarchies as in [33]. Poon and Domingos developed the OntoUSP system that automatically induces ontology from dependency-parsed texts via hierarchical clustering [91]. Linguistic approaches are also applied, such as OntoLearn [105] and TextToOnto [96]. All these methods mainly focus on unstructured texts, while we fully exploited the rich (semi)structured information available on the Web, such as infoboxes in Wikipedia, to help ontology construction. These two methods can benefit each other by either improving precision or extending coverage.

3.8 Conclusion

Wikipedia is developing as the authoritative store of human knowledge. Recently, the combined efforts of human volunteers have extracted numerous facts from Wikipedia, storing them as machine-readable object-attribute-value triples in Wikipedia infoboxes. Furthermore, machine-learning systems, such as our KYLIN, can use these infoboxes as training data, and then accurately extract even more triples from Wikipedia's natural-language text. This huge repository of structured data could enable next-generation question answering systems which allow SQL-like queries over Wikipedia data, faceted browsing, and other capabilities. However, in order to realize the full power of this information, it must be situated in a cleanly-structured ontology.

This chapter makes a step in this direction, presenting KOG, an autonomous system for generating Wikipedia's ontology. We cast the problem of ontology creation as a machine learning problem and present a novel solution based on joint inference implemented using Markov Logic Networks. The resulting ontology contains subsumption relations and schema mappings between infobox classes; additionally, it maps these classes to WordNet.

Chapter 4

MOVING DOWN THE LONG TAIL

In Chapter 2, we presented KYLIN — a self-supervised system for information extraction from Wikipedia. KYLIN looks for sets of pages with similar infoboxes, determines common attributes for each class, creates training examples, learns extractors, and runs them on each page — creating new infoboxes and completing others.

KYLIN works extremely well for popular infobox classes where users have previously created sufficient infoboxes to train an effective extractor model. For example, in the "U.S. County" class KYLIN has 97.3% precision with 95.9% recall. Unfortunately, however, many classes (e.g. "Irish Newspaper") contain only a *small number* of infobox-containing articles. As shown in Figure 4.1, 1442 of 1756 (82%) classes have fewer than 100 instances, and 42% have 10 or fewer instances[1]. For classes sitting on this long tail, KYLIN can't get enough training data — hence its extraction performance is often unsatisfactory for these classes.

Furthermore, even when KYLIN does learn an effective extractor there are numerous cases where Wikipedia has an article on a topic, but the article simply doesn't have much information to be extracted. Indeed, another long-tailed distribution governs the *length* of articles in Wikipedia; among the 1.8 million pages, many are short articles and almost 800,000 (44.2%) are marked as *stub* pages, indicating that much-needed information is missing.

An ideal machine reading system should conquer both head and long tail textual knowledge. In order to create a comprehensive semantic knowledge base summarizing the topics in Wikipedia, we must confront the problems entailed by these long-tailed distributions. We must train extractors to operate on sparsely populated infobox classes and we must resort to other information sources if a Wikipedia article is superficial.

---

[1]Unless noted otherwise, all statistics in this chapter are taken from the 07/16/2007 snapshot of Wikipedia's English language version.

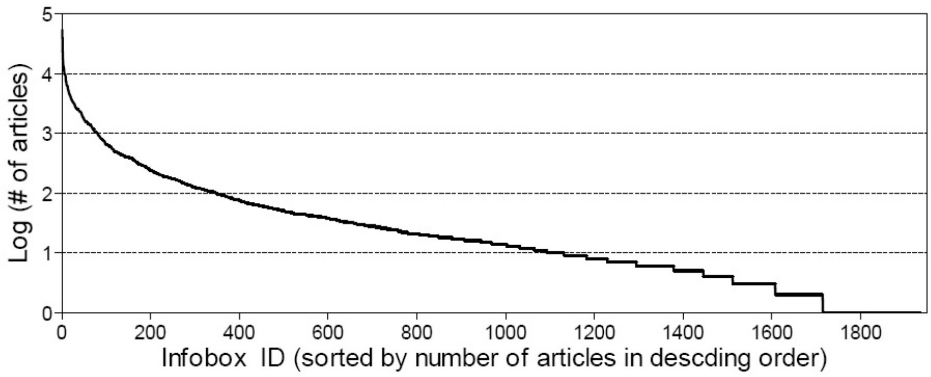


Figure 4.1: The number of article instances per infobox class has a long-tailed distribution.

In this chapter we describe three novel approaches for improving recall of KYLIN's information extraction on Wikipedia.

- By applying shrinkage [73, 102] over an automatically-learned subsumption taxonomy, we allow KYLIN to substantially improve the recall of its extractors for sparsely populated infobox classes.

- By mapping the contents of known Wikipedia infobox data to TextRunner, a state-of-the-art open information extraction system [15], we enable KYLIN to clean and augment its training dataset. When applied in conjunction with shrinkage, this *re-training* technique improves recall by a factor of between 1.4 and 5.9, depending on class.

- When it is unable to extract necessary information from a Wikipedia page, we enable KYLIN to retrieve relevant sentences from the greater Web. As long as tight filtering is applied to non-Wikipedia sources, recall can be still further improved while maintaining high precision.

Our techniques work best in concert. Together, they improve recall by a factor of 1.89 to 8.42 while maintaining or increasing precision. The area under the precision-recall curve increases by a factor of between 1.91 to 8.71, depending on class. In addition to showing the

great cumulative effect of these techniques, we analyze several variations of each method, exposing important engineering tradeoffs. We describe each technique in the next sections.

4.1 Shrinkage

Although KYLIN performs well when it can find enough training data, it flounders on sparsely populated infobox classes — the majority of cases. Our first attempt to improve KYLIN's performance uses shrinkage, a general statistical technique for improving estimators in the case of limited training data [102]. McCallum *et al.* applied this technique for text classification in a hierarchy classes by smoothing parameter estimate of a data-sparse child with its parent to get more robust estimates [73].

Similarly, we use shrinkage when training an extractor of an instance-sparse infobox class by aggregating data from its parent and children classes. For example, knowing that *performer ISA person*, and *performer.loc=person.birth_plc*, we can use values from *person.birth_plc* to help train an extractor for *performer.loc*. The trick is automatically generating a good subsumption hierarchy which relates attributes between parent and child classes. Fortunately, the ontology produced by our KOG system (described in Chapter 3) perfectly meets this requirement.

Given a sparse target infobox class C, KYLIN's shrinkage module searchs upwards and downwards through the KOG ontology to aggregate training data from related classes. The two crucial questions are: 1) How far should one traverse the tree? 2) What should be the relative weight of examples in the related class compared to those in C? For the first question, we search to a uniform distance, l, outward from C. In answer to the second question, we evaluate several alternative weighting schemes in Section 4.1.1. The overall shrinkage procedure is as follows:

1. Given a class C, query KOG to collect the related class set: $S_C = \{C_i | path(C, C_i) \leq l\}$, where l is the preset threshold for path length. Currently KYLIN only searches strict parent/child paths without considering siblings. Take the "Performer" class as an example: its parent "Person" and children "Actor" and "Comedian" could be included in S_C.

2. For each attribute $C.a$ (e.g. "Performer.birthplace") of C:

   (a) Query KOG for the mapped attribute $C_i.a_j$ (e.g. "Person.cityofbirth") for each $C_i \in S_C$.

   (b) Assign weight w_{ij} to the training data from $C_i.a_j$ and aggregate them in the training dataset for $C.a$. Note that w_{ij} may be a function of the target attribute a, the related class C_i, and C_i's mapped attribute a_j.

3. Train the CRF extractors for C based on the aggregated training dataset.

4.1.1 Shrinkage Experiments

This section addresses two questions:

- Does shrinkage over the KOG ontology help KYLIN to learn extractors for sparse classes? What if the target class is *not* sparse?

- What is the best strategy for computing the training weights, w_{ij}?

To answer these questions we used the `07/16/2007` snapshot of `en.wikipedia.org` as a source dataset. We tested on four classes, namely "Irish Newspaper" (which had 20 infobox-contained instance articles), "performer" (44), "baseball stadium" (163), and "writer" (2213). These classes represent various degrees of "sparsity" in order to provide better understanding of how shrinkage helps in different cases. For the "Irish Newspaper" and "performer" classes, we manually labeled all the instances to compute precision and recall values. Particularly, we count the ground-truth as the attribute values contained in the articles — meaning a 100 percent recall is getting every attribute value which is present in the article. For the "baseball stadium" and "writer" classes, we manually labeled 40 randomly-selected instances from each. All the following experiments use 4-cross-validations.

After schema cleaning, KOG identified 1269 infobox classes and mapped them to the WordNet lattice (82115 synsets). We found that although the whole ontology is quite dense, the current number of Wikipedia infoboxes is relatively small and most pathes through the

Target class	Parent	Children
Irish Newspaper(20)	Newspaper(1559)	–
Performer(44)	Person(1201)	Actor(8738) Comedian(106)
Baseball stadium(163)	Stadium(1642)	–
Writer(2213)	Person(1201)	Sci-fi writer(36)

Table 4.1: Parent/children classes for shrinkage.

taxonomy cover three or fewer infobox classes, which diminishes the effect of path-length threshold l. Table 4.1 shows the detailed parent/children classes for each testing case. In the following, we mainly focus on testing weighting strategies.

We considered three strategies to determine the weights w_{ij} for aggregated data from parent/child classes:

Uniform: $w_{ij} = 1$, which weights all training samples equally.

Size Adjusted: $w_{ij} = min\{1, \frac{k}{|D(C)|+1}\}$, where k (10 in our experiments) is the design parameter, and $|D(C)|$ is the number of instance articles contained in C. The intuition is that the bigger $D(C)$ is, the less shrinkage should rely on other classes.

Precision Directed: $w_{ij} = p_{ij}$, where p_{ij} is the extraction precision when applying C_i's extractor on the appropriate sentences from C-class articles and comparing them with existing infobox values.

Even with limited parent/children classes for smoothing, all forms of shrinkage improve extraction performance. Figure 4.2 shows the precision/recall curves for our different weighting strategies. We draw several conclusions:

First, with shrinkage, KYLIN learns better extractors, especially in terms of recall. For those very sparse classes such as "performer" and "Irish Newspaper", the recall improvement is dramatic: 57% and 460% respectively; and the area under the precision and recall curve improves 62% and 1420% respectively.

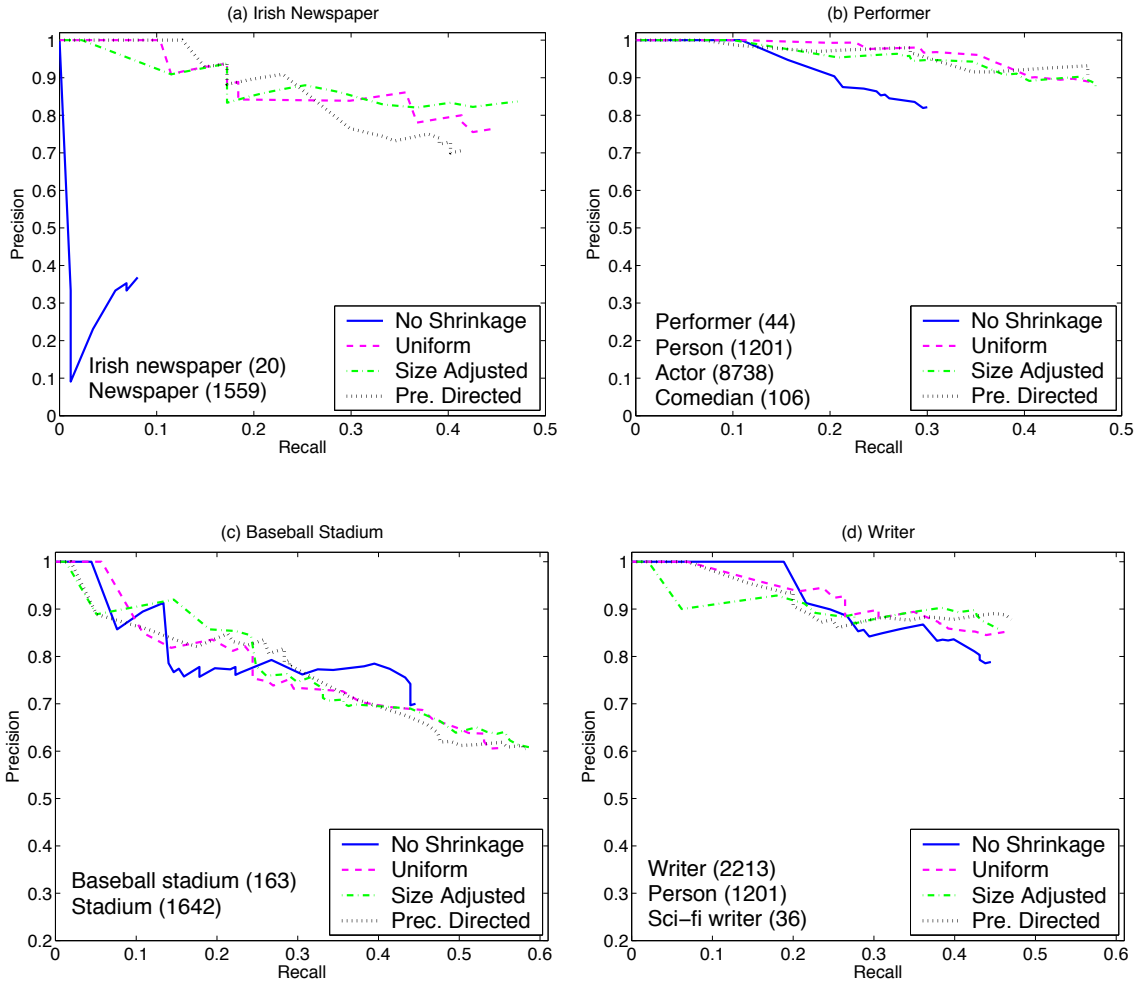


Figure 4.2: Regardless of the weighting scheme, extractors trained with KOG-enabled shrinkage outperforms the KYLIN baseline — especially on the sparse "Irish Newspaper," "Performer" and "Baseball Stadium" classes where recall is dramatically improved. In the two sparsest classes, precision is also markedly improved.

Second, we expected precision-directed shrinkage to outperform the other methods of weighting, since it automatically adapt to different degrees of similarity between the targent and related classes. However, the three weighting strategies turn out to perform comparatively on the infobox classes used for testing. The most likely reason is that to achieve total autonomy KYLIN estimates the precision, p_{ij}, of an extractor by comparing the values which it extracts to those entered manually in existing infoboxes. It turns out that in many cases Wikipedia editors use different expressions to describe attribute values in the infoboxes than they do in the article text. Naturally, this makes the accurate estimation of p_{ij} extremely difficult. This, in turn, biases the quality of weighting. In the future, we hope to investigate more sophisticated weighting methods.

Finally, Shrinkage also helps the quality of extraction in popular classes (e.g., for "writer"), though the improvement is quite modest. This is encouraging, since "writer" already had over two thousand training examples.

4.2 Retraining

Our experiments show that shrinkage enables KYLIN to find extra data *within* Wikipedia to help train extractors for sparse classes. A complementary idea is the notion of harvesting additional training data even from the *outside* Web? Leveraging information outside Wikipedia, could dramaticaly improve KYLIN's recall. To see why, we note that the wording of texts from the greater Web are more diverse than the relatively strict expressions used in many places in Wikipeidia.[2] Training on a wider variety of sentences would improve the robustness of KYLIN's extractors, which would potentially improve the recall.

The trick here is determining how to automatically identify relevant sentences given the sea of Web data. For this purpose, KYLIN utilizes TextRunner, an open information extraction system [15], which extracts semantic relations $\{r|r = \langle obj_1,\ predicate,\ obj_2\rangle\}$ from a crawl of about 500 million Web pages. Importantly for our purposes, Textrunner's crawl includes the top ten pages returned by Google when queried on the title of every Wikipedia article. In the next subsection, we explain the details of our retraining process; then we

[2]It is possible that Wikipedia's inbred style stems from a pattern where one article is copied and modified to form another. A general desire for stylistic consistency is another explanation.

follow with an experimental evaluation.

4.2.1 Using TextRunner for Retraining

Recall that each Wikipedia infobox implicitly defines a set of triples $\{t|t = \langle subject,\ attribute,\ value\rangle\}$ where the subject corresponds to the entity which is the article's title. These triples have the same underlying schema as the semantic relations extracted by TextRunner and this allows us to generate new training data.

The retrainer iterates through each infobox class C and again through each attribute, $C.a$, of that class collecting a set of triples from existing infoboxes: $T = \{t|t.attribute = C.a\}$.[3] The retrainer next iterates through T, issuing TextRunner queries to get a set of potential matches $R(C.a) = \{r|\exists t : r.obj_1 = t.subject, r.obj_2 = t.value\}$, together with the corresponding sentences which were used by TextRunner for extraction. The KYLIN retrainer uses this mapped set $R(C.a)$ to augment and clean the training data set for C's extractors in two ways: by providing additional positive examples for the learner, and by eliminating false negative examples which were mistakenly generated by KYLIN from the Wikipedia data.

ADDING POSITIVE EXAMPLES: Unfortunately, TextRunner's raw mappings, $R(C.a)$, are too noisy to be used as positive training examples. There are two causes for the noise. The most obvious cause is the imperfect precision of TextRunner's extractor. But false positive examples can also be generated when there are multiple interpretations for a query. Consider the TextRunner query $\langle r.obj_1 = A, r.predicate = ?, r.obj_2 = B\rangle$, where A is a person and B is his birthplace. Since many people die in the same place that they were born, TextRunner may well return the sentence "Bob died in Seattle." which would be a poor training example for birthplace.

Since false positives could greatly impair training, the KYLIN retrainer morphologically clusters the predicates which are returned by TextRunner (e.g., "is married to" and "was

[3] We note that another way of generating the set, T, would be to collect baseline KYLIN extractions for $C.a$ instead of using existing infoboxes. This would lead to a *cotraining* approach rather than simple retraining. One could iterate the process of getting more training date from TextRunner with improvements to the KYLIN extractor [21].

married to" are grouped). We discard any predicate that is returned in response to a query about more than one infobox attribute. Only the k most common remaining predicates are then used for positive training examples; in our experiments we set $k = 1$ to ensure high precision.

FILTERING FALSE NEGATIVE EXAMPLES: As explained in Section 2.2.3, KYLIN considers a sentence to be a negative example unless it is known to be positive or the *sentence classifier* labels it as potentially positive. This approach eliminates many false negatives, but some remain. A natural idea is to remove a sentence from the set of negative examples if it contains the word denoting the relation itself. Unfortunately, this technique is ineffective if based soley on Wikipedia content. To see why, consider the "Person.spouse" attribute which denotes the "marriage" relation —because the word "spouse" seldom appears in natural sentences, few false negatives are excluded. But by using TextRunner, we can better identify the phrases (predicates) which are harbingers of the relation in question. The most common are used to eliminate negative examples.

By adding new positive examples and excluding sentences which might be false negatives, retraining generates an improved training set. The next subsection shows the benefits of this approach.

4.2.2 Retraining Experiments

This section answers two questions:

- Does retraining improve KYLIN's extractors?
- Do the benefits from retraining combine synergistically with those from shrinkage?

Before addressing those questions we experimented with different retraining alternatives (e.g., just adding positive examples and just filtering negatives). While both approaches improved extractor performance, the combination worked best, so the combined method was used in the subsequent study.

We evaluate retraining in two different cases. In the first case, we use nothing but the target class' infobox data to prime TextRunner for training data. In the second case, we first used uniform-weight shrinkage to create a training set which was then used to query TextRunner. Figure 4.3 shows the results of these methods on four testing classes.

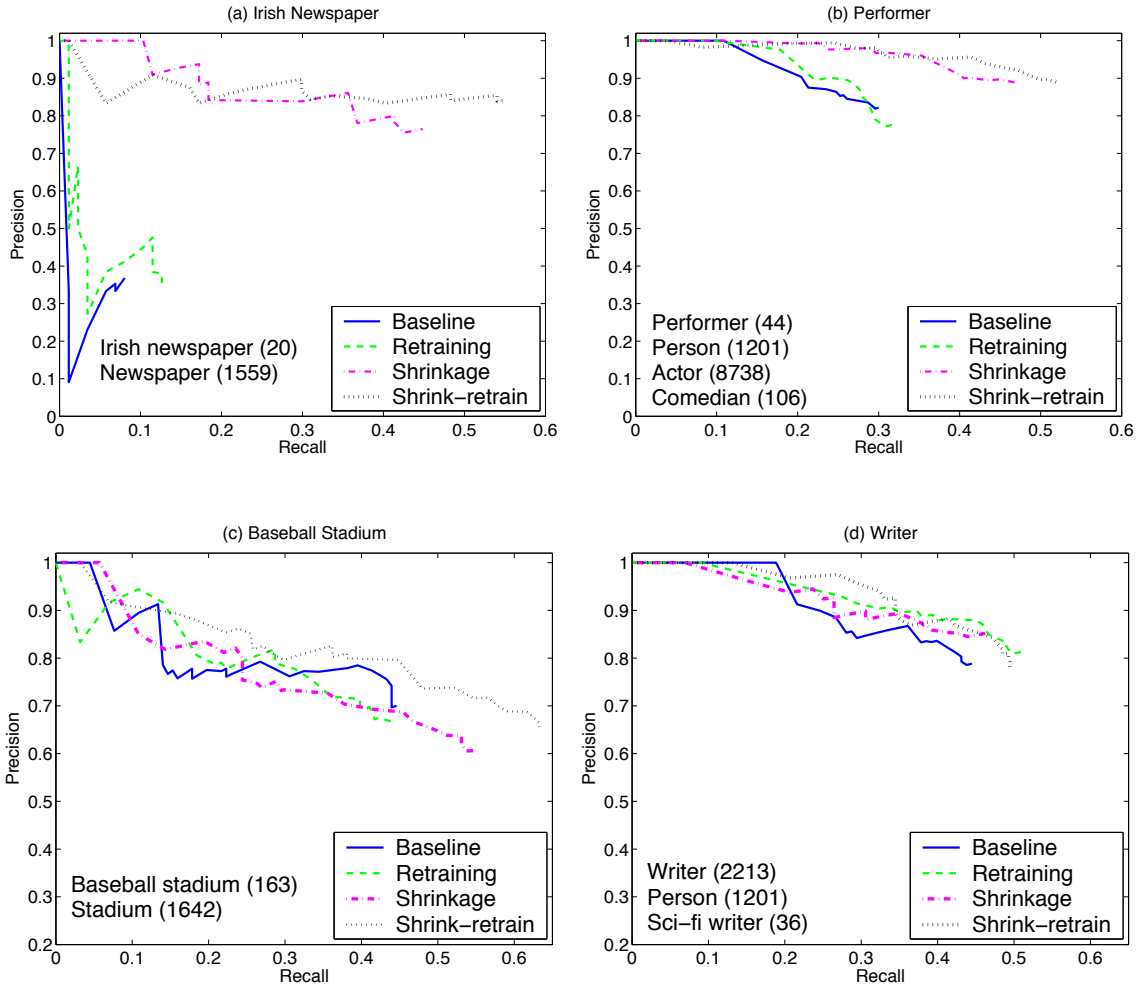


Figure 4.3: Used in isolation, retraining enables a modest but marked improvement in recall. And combining retraining with shrinkage yields substantially improved extractors with improvements to precision as well as recall.

We note that in most cases retraining improves the performance, in both precision and recall. When compared with shrinkage, retraining provides less benefit for sparse classes

but helps more on the popular class "writer." This makes sense because without many tuples to use for querying TextRunner, retraining has little effect. We suspect that full cotraining would be more effective on sparse classes when shrinkage was unavailable. Finally, we observe that the combination of shrinkage and retraining is synergistic, always leading to the biggest improvement. Particularly, on the two sparsest classes "Irish Newspaper" and "performer", it substantially improved recall by 590% and 73.3% respectively, with remarkable improvement in precision as well; and the areas under the precision and recall curve improve 1816% and 66% respectively.

4.3 Extracting from the Web[4]

While shrinkage and retraining improve the quality of KYLIN's extractors, the lack of redundancy of Wikipedia's content makes it increasingly difficult to extract additional information. Facts that are stated using uncommon or ambiguous sentence structures hide from the extractors.

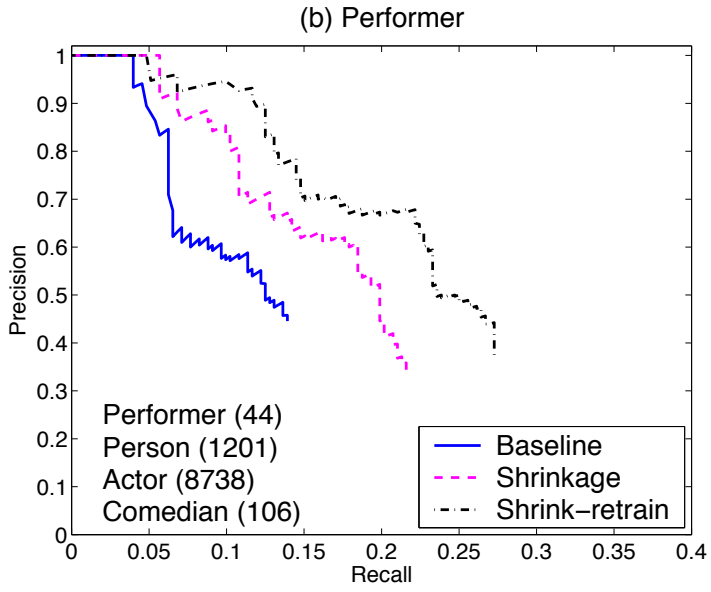


Figure 4.4: When applying KYLIN to Web pages, improvements due to shrinkage and retraining become even more apparent.

[4]Work in this subsection is done under collaboration with Raphael Hoffmann.

In order to retrieve facts which can't be extracted from Wikipedia, we would like to exploit another corpus, in particular the general Web. On the surface, the idea is simple: train extractors on Wikipedia articles and then apply them to relevant Web pages. An obvious benefit of this approach is the ability to find new facts which are not contained in Wikipedia at all.

The challenge for this approach — as one might expect — is maintaining high precision. Since the extractors have been trained on a very selective corpus, they are unlikely to discriminate irrelevant information. For example, a KYLIN extractor for a person's birthdate has been trained on a set of pages all of which have as their primary subject that person's life. Such extractors become inaccurate when applied to a page which compares the lives of several people — even if the person in question is one of those mentioned.

To ensure extraction quality, it is thus crucial to carefully select and weight content that is to be processed by KYLIN's extractors. In our work, we view this as an information retrieval problem, which KYLIN's web extraction module solves in the following steps: It generates a set of queries and utilizes a general Web search engine, namely Google, to identify a set of pages which are likely to contain the desired information. The top-k pages are then downloaded, and the text on each page is split into sentences, which are processed by KYLIN. Each extraction is then weighted using a combination of factors.

CHOOSING SEARCH ENGINE QUERIES: The first important step is to ensure that the search engine returns a set of highly relevant pages. A simple approach is to use the article title as a query. For example, let us assume that we are interested in finding the birth date of Andrew Murray, a writer. The corresponding Wikipedia page is titled 'Andrew Murray (minister)'. The information in parentheses is used in Wikipedia to resolve ambiguities, but we remove it to increase recall. To improve result relevance, we place quotes around the remaing string, here `‘‘andrew murray’’`.

Although such a query might retrieve many pages about Murray, it is possible that none among the top contains the person's birth date which we might be interested in. We therefore run several more restrictive queries which not only limit results to pages containing the article title, but that also include other keywords to better target the search.

One such query is the quoted article title followed by the attribute name, as in `‘‘andrew murray’’ birth date`. While this increases the chance that a returned page contains the desired information, it also greatly reduces recall, because the terms ‘birth date’ might not actually appear on a relevant page. For example, consider the sentence ‘Andrew Murray was born in 1828.”.

Such predicates which are indicative of attributes, like ‘was born in’ for the birth date, we have computed already, as described in section 4.2. We generate an appropriate query for each predicate, which combines the quoted title as well as the predicate, as in `‘‘andrew murray’’ was born in`. The combined results of all queries (title only, title and attribute name, as well as title and any attribute predicate) are retrieved for further processing.

WEIGHING EXTRACTIONS: Pages which do not contain the preprocessed article title, here ‘Andrew Murray’, are discarded. Then, using an HTML parser, formatting commands and scripts are removed, and sentences are identified in the remaining text.

Since most sentences are still irrelevant, running KYLIN’s extractors on these directly would result in many false positives. Recall that unlike Wikipedia’s articles, web pages often compare multiple related concepts, and so we would like to capture the likeliness that a sentence or extraction is relevant to the concept in question. A variety of features may be indicative of content relevance, but we focused on two in particular:

- The number of sentences δ_s between the current sentence and the closest sentence containing the (preprocessed) title of the article.

- The rank of the page δ_r on Google’s results lists returned in response to our queries.

Each retrieved sentence is then sent to KYLIN for extraction, and for each extraction a combined score is computed. This score takes into account both factors δ_s and δ_r as well as the confidence δ_c reported by KYLIN’s extractors; it is obtained in the following way: First, each of the three parameters $\delta_s, \delta_r, \delta_c$ is normalized by applying a linear mapping into the intervals $[\alpha_s, 1]$, $[\alpha_r, 1]$, and $[\alpha_c, 1]$ respectively, where 1 corresponds to the optimal value and α_s, α_r, and α_c are user-defined parameters. With δ_s^*, δ_r^*, and δ_c^* denoting the normalized weights, the combined score is then obtained as $score_{web} := \delta_s^* * \delta_r^* * \delta_c^*$.

COMBINING WIKIPEDIA AND WEB EXTRACTIONS: Our final question is: how can we combine extraction results from Wikipedia and the Web? Despite our efforts in identifying relevant Web pages and weighting sentences, it is likely that extractions from Wikipedia will be more precise. After all, in Wikipedia we can be sure that a given page is highly relevant, is of high quality, and has a more consistent structure, for which KYLIN's extractors have been particularly trained. Yet, KYLIN may err on Wikipedia too, especially when the extractors confidence score is low.

A straight-forward combination of the extractors is to always return the extraction with highest score, as measured in terms of confidence for extractions from Wikipedia and the weighted combination $score_{web}$ for extractions from the Web.

To be able to balance the weight of one extractor versus the other, we adjust the score of extractions from the web to $1 - (1 - score_{web})^{\lambda}$, where λ is a new parameter.

4.3.1 Web Experiments

In this section we would like to answer two questions:

- Which factors are important in scoring extractions from the Web?
- When combining extractions from Wikipedia and the Web, can recall be significantly improved at an acceptable precision?

In previous sections, we computed recall as the proportion of facts contained in the infoboxes that our system was able to automatically extract from the text. In this section, however, we are also interested in how many new facts KYLIN can extract from the Web, and so we change our definition of recall: we assume that there exists some correct value for each attribute contained in the infobox template of an article and set recall to be the proportion of correct attribute values relative to all attributes. Note that this is a very conservative estimate, since there may not always exist an appropriate value. For example, there exists no death date for a writer who has not died yet.

For all experiments, we queried Google for the top-100 pages containing the article title, and the top-10 pages containing the article title and the attribute name or any associated

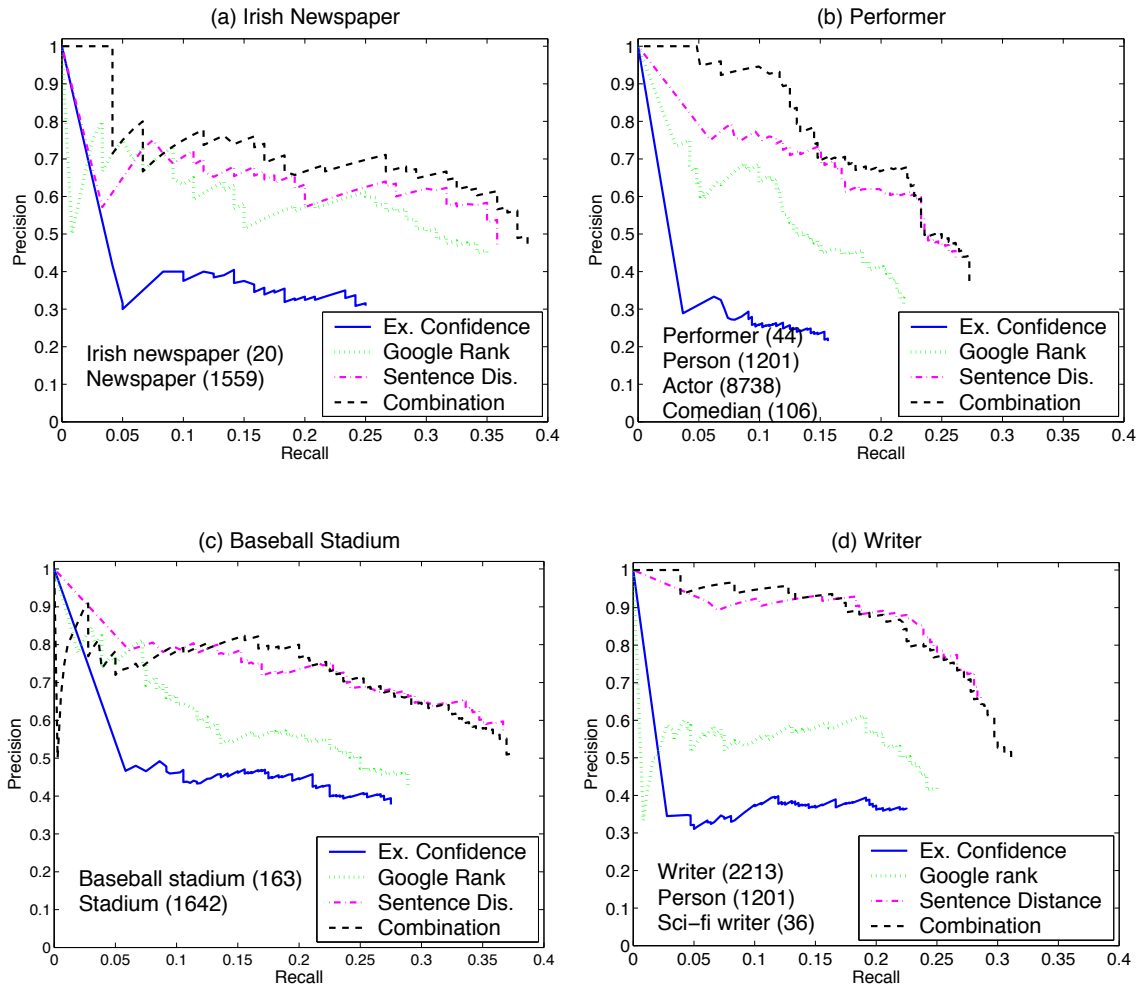


Figure 4.5: When applying KYLIN to Web pages, the CRF extractor's confidence is a poor choice for scoring competing extractions of the same attribute. Giving priority to extractions from pages ranked higher by Google, and resolving ties by extractor confidence, improves results considerably. 'Sentence Dis' which similarly gives priority to extractions from sentences which are closer to the next occurrence of the Wikipedia article title on a web page, improves further, and is only outperformed by a weighted combination of the other three factors.

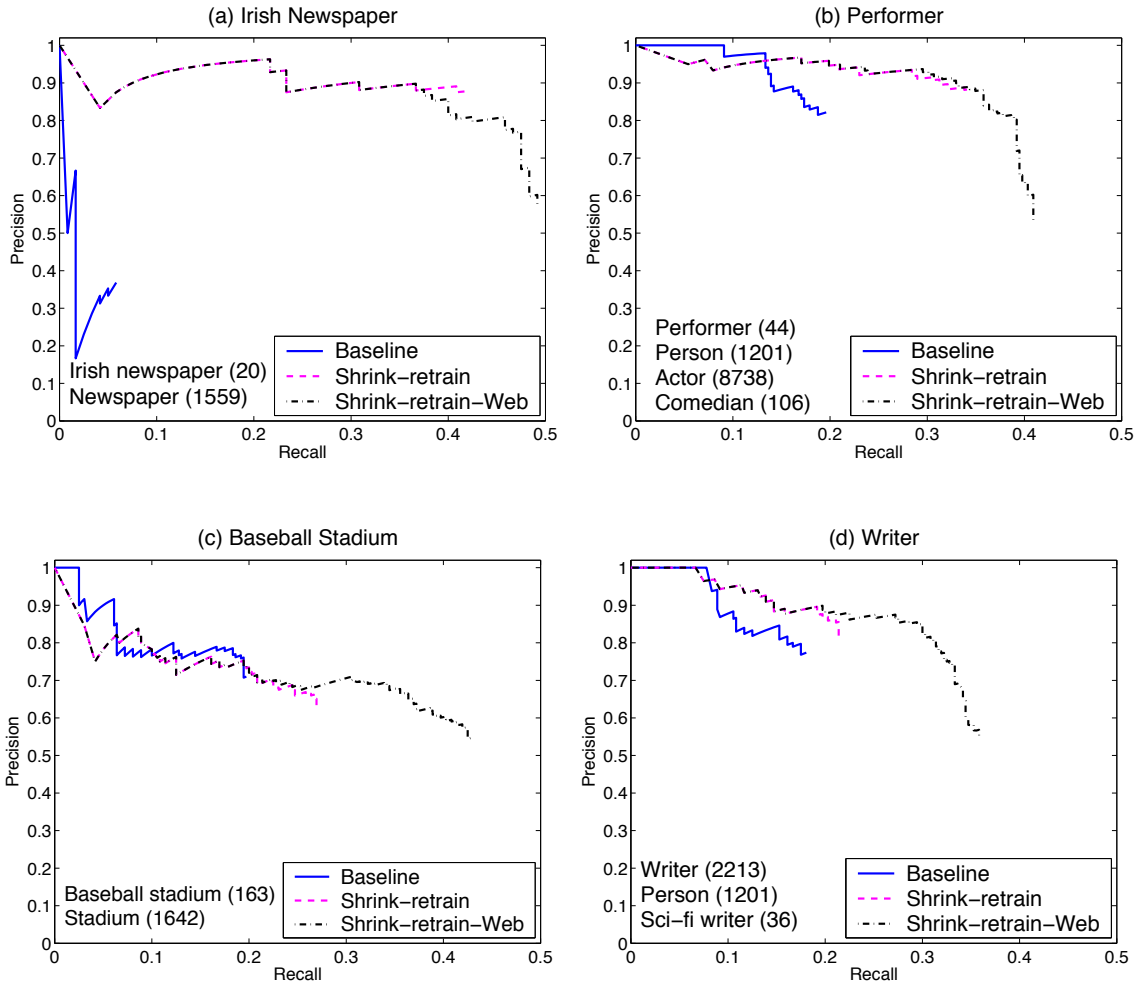


Figure 4.6: Combining KYLIN's extractions from Wikipedia and the Web yields a substantial improvement in recall without compromising precision. Already, shrink-retrain improved recall over the original KYLIN system, here the baseline, but the combination of extractions from Wikipedia and the Web, shrink-retrain-Web, performs even better.

predicate. Each new extraction — for which no ground truth existed in Wikipedia — was manually verified for correctness by visiting the source page.

In our first series of experiments, we used Shrink-Retrain — the best extractors trained on Wikipedia — and applied different scoring functions to select the best extraction for an attribute. Figure 4.5 shows our results: The CRF extractor's reported confidence performed poorly in isolation. Giving priority to extractions from pages at a higher position in Google's returned result lists and resolving ties by confidence, yielded a substantial improvement. Similarly, we tried giving priority to extractions which were fewer sentences apart from the occurrence of the Wikipedia article title on a page, again resolving ties by extractor confidence. The large improvements in precision and recall (as highlighted in the figure 4.5) show that much of the returned text is irrelevant, but can be re-weighted using simple heuristics. Finally, we were interested if a weighted combination of these factors would lead to synergies. We set $\alpha_s = .1$, $\alpha_r = .7$, $\alpha_c = .9$, so that each factor was roughly weighted by our observed improvement (results were not sensitive to minor variations). On all datasets, performance was comparable or better than the best factor taken in isolation.

In our second series of experiments, we combined extractions from Wikipedia and the Web. In both cases, we applied the Shrink-Retrain extractor, but scored extractions from the Web using the weighted factor combination with $\lambda = .4$. The results, shown in Figure 4.6, show large improvements in recall at higher precision for the Baseball Stadium (38%) and Writer (42%) datasets, and at moderately reduced precision for the Irish Newspaper and Performer datasets. The area under the curve was substantially expanded in all cases, ranging from 15% to 58%. Compared to the original baseline system, the area has expanded between 91% and 1771%.

In the future, we would like to automatically optimize the parameters α_s, α_r, α_c, λ based on comparing the extractions with values in the infobox.

4.4 Related Work

In the preceding sections we have discussed how our work relates to past work on shrinkage and cotraining. In this section, we discuss the broader context of previous work on unsupervised information extraction, and approaches for exploiting ontologies in information

extraction.

Unsupervised and Self-Supervised Information Extraction: Unsupervised and self-supervised learning is necessary for Web-scale IE. Patwardhan and Riloff proposed a decoupled information extraction system by first creating a self-trained relevant sentence classifier to identify relevant regions, and using a semantic affinity measure to automatically learn domain-relevant extraction patterns [85]. KYLIN uses the similar idea of decoupling when applying extractors to the general Web. Differently, KYLIN uses IR-based techniques to select relevant sentences and trains a CRF model for extractions. For more discussion on unsupervised and self-supervised IE, please refer to Section 2.4 (Related Work).

Ontology-Driven Information Extraction: There have been a lot of work on leveraging ontology for information extraction. The SemTag and Seeker [44] systems perform automated semantic tagging of large corpora. They use the TAP knowledge base [94] as the standard ontology, and match it with instances on the Web. PANKOW [32] queries Google with ontology-based Hearst patterns to annotate named entities in documents. Matuszek *et al.* uses Cyc to specify Web searches to identify and verify common senses candidates [72]. The similar idea is utilized in OntoSyphon [75] where ontology combined with search engines are used to identify semantic instances and relations. In contrast, KYLIN automatically constructs the Wikipedia infobox ontology and uses it to help training CRF extractors by shrinkage.

4.5 Conclusion

KYLIN has demonstrated the ability to perform self-supervised information extraction from Wikipedia. While KYLIN achieved high precision and reasonable recall when infobox classes had a large number of instances, most classes (i.e., 82%) can provide fewer than 100 training examples for these classes, where KYLIN's performance is unacceptable.

This chapter describes three powerful methods for increasing recall in sparsely populated classes: shrinkage, retraining, and supplementing Wikipedia extractions with those from the Web. Our experiments show that each of these methods is effective individually. We

evaluate design tradeoffs within each method. Most importantly, we show that in concert, these methods constitute a huge improvement to KYLIN's performance (Figure 4.6):

- Precision is modestly improved in most classes, with larger gains if sparsity is extreme (e.g., "Irish Newspaper").

- Recall sees extraordinary improvement with gains from 0.06% to 0.49% (a factor of 8.4) in extremely sparse classes such as "Irish Newspaper." Even though the "Writer" class is populated with over 2000 infoboxes, its recall improves from 18% to 32% (a factor of 1.77) at equivalent levels of precision.

- Calculating the area under the precision / recall curve also demonstrates substantial improvement, with an improvement factor of 8.71, 2.02, 1.91, and 1.93 for "Irish Newspaper," "Performer," "Baseball Stadium," and "Writer," respectively.

Despite this success, much remains to be done. We hope to devise a better weighting scheme for shrinkage by comparing the KL divergence between the target and mapped classes. We wish to extend our retraining technique to full cotraining. There are several ways to better integrate extraction of Web content with that of Wikipedia, ranging from improved Google querying policies to DIRT-style analysis of extraction patterns [68].

Chapter 5

WOE: OPEN INFORMATION EXTRACTION USING WIKIPEDIA

The problem of information-extraction (IE), generating relational data from natural-language text, has received increasing attention in recent years. The vast majority of IE work uses supervised learning of relation-specific examples. For example, the WebKB project [37] used labeled examples of the `courses-taught-by` relation to induce rules for identifying additional instances of the relation. While these methods can achieve high precision and recall, they are limited by the availability of training data. In Chapter 2 we developed the KYLIN system which matches Wikipedia infoboxes with sentences to automatically label a few million training examples for learning a broad set of relation-specific extractors. KYLIN achieves a big step toward machine reading on the Web. However, it is limited by the scope of relations defined in Wikipedia infoboxes, and is incapable of handling an unbounded number of relations embedded in Web documents.

An alternative paradigm, *Open* IE, pioneered by the TextRunner system [14] and "preemptive IE" in [98], aims to handle an unbounded number of relations and run quickly enough to process Web-scale corpora. Domain independence is achieved by extracting the relation name as well as its two arguments. Most open IE systems use self-supervised learning, in which automatic heuristics generate labeled data for training the extractor. For example, TextRunner uses a small set of hand-written rules to heuristically label training examples from sentences in the Penn Treebank.

This chapter presents WOE (Wikipedia-Based Open Extractor), the first system that autonomously transfers knowledge from random editors' effort of collaboratively editing Wikipedia, to train an open information extractor. Specifically, WOE generates *relation-specific* training examples by matching infobox attribute values to corresponding sentences (as done in KYLIN and Luchs [61]), but WOE abstracts these examples to *relation-independent* training data to learn an unlexicalized extractor, akin to that of TextRunner. WOE can

operate in two modes: when restricted to shallow features like part-of-speech (POS) tags, it learns a second-order linear chain CRF extractor and runs as quickly as TEXTRUNNER — 0.022 seconds per sentence in average; but when set to use parse features, it learns a pattern classifier based on shortest-dependency-path features whose precision and recall rise even higher. However, this performance improvement comes at the cost of speed: it takes 0.679 seconds to process one sentence — 30X times slower.

5.1 Open Information Extraction

In order to extract the widely-varying type of information on the Web, attention has recently turned to the broader goal of what Etzioni *et al.* call *open* information extraction [14, 52] — the task of scalably extracting information to fill an unbounded number of relational schemata, whose structure is unknown in advance. Open IE is distinguished from traditional methods on three dimensions [52]:

- **Input:** Traditional, supervised approaches require a set of labeled training data in addition to the corpus for extraction; open IE uses domain-independent methods instead.

- **Target Schema:** In traditional IE, the target relation is specified in advance; open IE automatically discovers the relations of interest.

- **Computational Complexity:** The runtime of traditional methods is $O(D * R)$, where D denotes the number of documents in the corpus and R denotes the number of relations; in contrast, scalability to the Web demands that open IE scale linearly in D.

In this work we focus on learning an open extractor to render a document, d, into a set of triples, $\{\langle \texttt{arg}_1, \texttt{rel}, \texttt{arg}_2 \rangle\}$, where the `arg`s are noun phrases and `rel` is a textual fragment indicating an implicit, semantic relation between the two noun phrases. The extractor should produce one triple for every relation stated *explicitly* in the text, but is not required to infer implicit facts. It is subjective to evaluate the correctness of an

extracted triple. Taking the sentence "Joe received a PhD degree in CS" for example, $\langle Joe, received, a\ PhD\ degree\ in\ CS\rangle$ and $\langle Joe, received\ a\ PhD\ degree\ in, CS\rangle$ are two potential extractions. Although more users might prefer $\langle Joe, received, a\ PhD\ degree\ in\ CS\rangle$ as a better triple, it's arguable that $\langle Joe, received\ a\ PhD\ degree\ in, CS\rangle$ is also a reasonable one. In our experiment, we use Amazon Mechanical Turk to verify the correct tuples from test sentences to reduce the labeling bias. In this work, we assume that all relational instances are stated within a single sentence. Note the difference between open IE and the traditional approaches (*e.g.*, as in WebKB), where the task is to decide whether some pre-defined relation holds between (two) arguments in the sentence.

We develop the WOE system to learn an open extractor *without direct supervision*, *i.e.* without annotated training examples or hand-crafted patterns. The input to WOE is Wikipedia[1]. As output, WOE produces an unlexicalized and relation-independent open extractor, which generalizes beyond Wikipedia, handling other corpora such as the general Web. We describe the components and operations of WOE in the following section.

5.2 Wikipedia-Based Open Information Extraction

The key idea underlying WOE is the automatic construction of training examples by heuristically matching Wikipedia infobox values and corresponding text, as done in KYLIN; these examples are then abstracted and used to generate an unlexicalized, relation-independent (open) extractor. As shown in Figure 5.1, WOE has three main components: preprocessor, matcher, and learner. We describe each module in more details in the following sections.

5.2.1 Preprocessor

The preprocessor converts the raw Wikipedia text into a sequence of sentences, attaches NLP annotations, and builds synonym sets for key entities. Figure 5.2 shows the compact pseudocode. The resulting data is fed to the matcher, described in Section 5.2.2, which generates the training set.

---

[1]We also use DBpedia [12] as a collection of conveniently parsed Wikipedia *infoboxes*

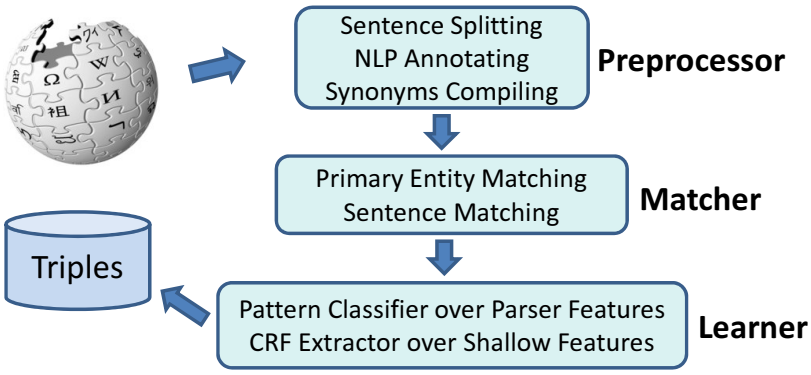


Figure 5.1: Architecture of WOE.

Sentence Splitting: The preprocessor first renders each Wikipedia article into HTML, then splits the article into sentences using OpenNLP, as done in KYLIN.

NLP Annotation: As we discuss fully in Section 5.3 (Experiments), we consider several variations of our system; one version, WOE^{parse}, uses parser-based features, while another, WOE^{pos}, uses shallow features like POS tags, which may be more quickly computed. Depending on which version is being trained, the preprocessor uses OpenNLP to supply POS tags and NP-chunk annotations — or uses the Stanford Parser to create a dependency parse. When parsing, we force the hyperlinked anchor texts to be a single token by connecting the words with an underscore; this transformation improves parsing performance in many cases.

Compiling Synonyms: As a final step, the preprocessor builds sets of synonyms to help the matcher find sentences that correspond to infobox relations. This is useful because Wikipedia editors frequently use multiple names for an entity; for example, in the article titled "University of Washington" the token "UW" is widely used to refer the university. Additionally, attribute values are often described differently within the infobox than they are in surrounding text. Without knowledge of these synonyms, it is impossible to construct good matches. Following [79, 108], the preprocessor uses Wikipedia redirection pages and backward links to automatically construct synonym sets. Redirection pages are a natural choice, because they explicitly encode synonyms; for example, "USA" is redirected to the article on the "United States." Backward links for a Wikipedia entity such as the "Mas-

Preprocessor in WOE (Wikipedia Corpus):
SENTS = {}
1.Sentence Splitting:
  For each Wikipedia article d
    Split d into sentences with OpenNLP to get the sentence set SENT(d)={s_i|s_i in d}
    Add <d, SENT(d)> to SENTS
2. NLP Annotation:
  For each s_i in each SENT(d)
    Annotate s_i with NLP tags (POS, NP-chunk, Parsing), denoting as NLP(s_i)
    Update SENT(d) to SENT(d)={<s_i, NLP(s_i)>|s_i in d}
3. Synonyms Compiling:
  S_R={}, S_B={}
  For each primary entity e in Wikipedia
    Get synonyms of e based on redirection pages: $S_r(e)$={e_i|e_i ~ e}
    Get synonyms of e based on backward-links: $S_b(e)$={e_i|e_i ~ e}
    Add $S_r(e)$ to S_R, add $S_b(e)$ to S_B
Return SENTS, S_R, S_B

Figure 5.2: WOE's preprocessor converts the raw Wikipedia text into a sequence of sentences, attaches NLP annotations, and builds synonym sets for key entities.

sachusetts Institute of Technology" are hyperlinks pointing to this entity from other articles; the anchor text of such links (*e.g.*, "MIT") forms another source of synonyms.

5.2.2 Matcher

The matcher constructs a set of training examples for the learner component by heuristically matching Wikipedia infoboxes with sentences, as done in KYLIN (Section 2.2.1). However, since an open extractor should be able to crop both arg_1 and arg_2 from sentences, the matcher needs to identify both primary entities (arg_1) and attribute values (arg_2, as done in KYLIN) in sentences. Figure 5.3 shows the compact pseudocode of the matcher.

Matching Primary Entities: In order to match shorthand terms like "MIT" with more complete names, the matcher uses an ordered set of heuristics like those of [80, 108]:

1. **Full match:** strings matching the full name of the entity are selected.

Matcher in WOE (annotated Wikipedia article <d, SENT(d)>):
TD = {}
1.Match Primary Entity:
    For each sentence s_i in d
        If successfully locate the substring arg_1 denoting the primary entity of d
            Add <s_i, arg_1> to TD
2. Match Sentence:
    For each infobox attribute C.a_i in d
        If successfully locate a unique s_i mentioning the attribute value of C.a_i in d
            Denote the attribute value in s_i as arg_2
            Update <s_i, arg_1> to <s_i, arg_1, arg_2> in TD
    Delete all <s_i, arg_1> in TD
Return TD

Figure 5.3: WOE's matcher constructs a set of training examples by heuristically identifying both primary entities and infobox attribute values in sentences.

2. **Synonym set match:** strings appearing in the entity's synonym set are selected.

3. **Partial match:** strings matching a prefix or suffix of the entity's name are selected. If the full name contains punctuation, only a prefix is allowed. For example, "Amherst" matches "Amherst, Mass," but "Mass" does not.

4. **Patterns of "the <*type*>":** The matcher first identifies the type of the entity (*e.g.*, "city" for "Ithaca"), then instantiates the pattern to create the string "the city." Since the first sentence of most Wikipedia articles is stylized (*e.g.* "The city of Ithaca sits ..."), a few patterns suffice to extract most entity types.

5. **The most frequent pronoun:** The matcher assumes that the article's most frequent pronoun denotes the primary entity, *e.g.*, "he" for the page on "Albert Einstein." This heuristic is dropped when "it" is most common, because the word is used in too many other ways.

When there are multiple matches to the primary entity in a sentence, the matcher picks the one which is closest to the matched attribute value in the parser dependency graph.

Matching Sentences: The matcher seeks a *unique* sentence to match the attribute value as done in KYLIN. To produce the best training set, the matcher performs three more filterings than KYLIN. First, it skips the attribute completely when multiple sentences mention the value or its synonym. Second, it rejects the sentence if the subject and/or attribute value are not heads of the noun phrases containing them. Third, it discards the sentence if the subject and the attribute value do not appear in the same clause (or in parent/child clauses) in the parse tree.

Since Wikipedia's Wikimarkup language is semantically ambiguous, parsing infoboxes is surprisingly complex. Fortunately, DBpedia [12] provides a cleaned set of infoboxes from 1,027,744 articles. The matcher uses this data for attribute values, generating a training dataset with a total of 301,962 labeled sentences.

5.2.3 Learning Extractors

We learn two kinds of extractors, one (WOE^{parse}) using features from dependency-parse trees and the other (WOE^{pos}) limited to shallow features like POS tags. WOE^{parse} uses a pattern learner to classify whether the shortest dependency path between two noun phrases indicates a semantic relation. In contrast, WOE^{pos} (like TEXTRUNNER) trains a conditional random fields (CRF) model to output certain text between noun phrases when the text denotes such a relation. Neither extractor uses individual words or lexical information for features. Figure 5.4 shows the compact pseudocode of the learner in WOE.

Extraction with Parser Features

Despite some evidence that parser-based features have limited utility in IE [63], we hoped dependency paths would improve precision on long sentences.

Shortest Dependency Path as Relation: Unless otherwise noted, WOE uses the Stanford Parser to create dependencies in the "collapsedDependency" format. Dependencies involving prepositions, conjuncts as well as information about the referent of relative clauses

Learner in WOE (training dataset from Matcher TD):
For each <s_i, arg_1, arg_2> in TD
  Get the *corePath* (shortest dependency path) between arg_1 and arg_2 in s_i
  Update <s_i, arg_1, arg_2> to <s_i, arg_1, arg_2, *corePath*>
1. WOE^{parse} - Extractor Based on Parser Features:
  DB_p = {}
  For each <s_i, arg_1, arg_2, *corePath*> in TD
    Convert *corePath* to *generalized-corePath, p*
    f_p++
    Add/Update <p, f_p> in DB_p
  Build a pattern classifier WOE^{parse} based on DB_p
2. WOE^{pos} - Extractor Based on Shallow Features:
  For each <s_i, arg_1, arg_2, *corePath*> in TD
    Convert *corePath* to *expandPath*
    Denote the tokens in *expandPath* as rel
    Update <s_i, arg_1, arg_2, *corePath*> to <s_i, arg_1, arg_2, *corePath*, rel>
  Learn a CRF extractor WOE^{pos} based on {<s_i, arg_1, arg_2, *corePath*, rel>}
Return WOE^{parse}, WOE^{pos}

Figure 5.4: WOElearns two kinds of extractors based on parser features and shallow features, respectively.

are collapsed to get direct dependencies between content words. As noted in [41], this collapsed format often yields simplified patterns which are useful for relation extraction. Consider the sentence:

Dan was not born in Berkeley.

The Stanford Parser dependencies are:

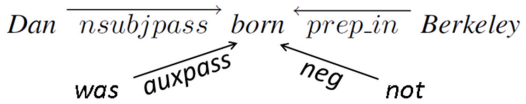
nsubjpass(born-4, Dan-1)
auxpass(born-4, was-2)
neg(born-4, not-3)
prep_in(born-4, Berkeley-6)

where each atomic formula represents a binary dependence from dependent token to the governor token.

These dependencies form a directed graph, $\langle V, E \rangle$, where each token is a vertex in V, and E is the set of dependencies. For any pair of tokens, such as "Dan" and "Berkeley", we use the shortest connecting path to represent the possible relation between them:

$$Dan \xrightarrow{nsubjpass} born \xleftarrow{prep\_in} Berkeley$$

We call such a path a *corePath*. While we will see that corePaths are useful for indicating *when* a relation exists between tokens, they don't necessarily capture the *semantics* of that relation. For example, the path shown above doesn't indicate the existence of negation! In order to capture the *meaning* of the relation, the learner augments the corePath into a tree by adding all adverbial and adjectival modifiers as well as dependencies like "neg" and "auxpass". We call the result an *expandPath* as shown below:



WOE traverses the expandPath with respect to the token orders in the original sentence when *outputting* the final expression of `rel`.

Building a Database of Patterns: For each of the 301,962 sentences selected and annotated by the matcher, the learner generates a corePath between the tokens denoting the subject and the infobox attribute value. Since we are interested in eventually extracting "subject, relation, object" triples, the learner rejects corePaths that don't start with subject-like dependencies, such as *nsubj*, *nsubjpass*, *partmod* and *rcmod*. This leads to a collection of 259,046 corePaths.

To combat data sparsity and improve learning performance, the learner further generalizes the corePaths in this set to create a smaller set of *generalized-corePaths*. The idea is to eliminate distinctions which are irrelevant for recognizing (domain-independent) relations. Lexical words in corePaths are replaced with their POS tags. Further, all Noun POS tags and "PRP" are abstracted to "N", all Verb POS tags to "V", all Adverb POS tags to "RB" and all Adjective POS tags to "J". The preposition dependencies such as "prep_in" are generalized to "prep". Take the corePath "$Dan \xrightarrow{nsubjpass} born \xleftarrow{prep\_in}$

Berkeley" for example, its generalized-corePath is "$N \xrightarrow{nsubjpass} V \xleftarrow{prep} N$". We call such a generalized-corePath an extraction pattern. In total, WOE builds a database (named DB_p[2]) of 15,333 distinct patterns and each pattern p is associated with a frequency — the number of matching sentences containing p. Figure 5.5 shows the logarithmic frequencies of the patterns. Specifically, 185 patterns have $f_p \geq 100$ and 1929 patterns have $f_p \geq 5$. Table 5.1 shows the top 20 extraction patterns.

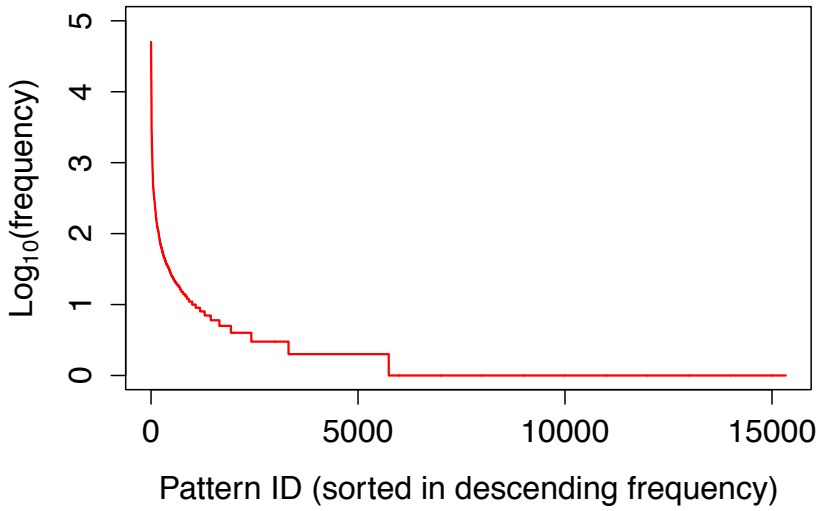


Figure 5.5: The frequency of extraction patterns has a long-tailed distribution.

Learning a Pattern Classifier: Given the large number of patterns in DB_p, we assume few valid open extraction patterns are left behind. The learner builds a simple pattern classifier, named WOE^{parse}, which checks whether the generalized-corePath from a test triple is present in DB_p, and computes the normalized logarithmic frequency as the probability[3]:

$$w(p) = \frac{max(log(f_p) - log(f_{min}), 0)}{log(f_{max}) - log(f_{min})} \tag{5.1}$$

[2]Available at: http://ai.cs.washington.edu/www/media/downloadable/media/patternDB-WOE.zip

[3]How to learn a more sophisticated weighting function is left as a future topic.

Pattern	Frequency
N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N	50259
N $\overrightarrow{nsubj}$ N $\overleftarrow{prep}$ N	32846
N $\overrightarrow{nsubjpass}$ V $\overleftarrow{prep}$ N	29112
N $\overrightarrow{nsubj}$ N	24199
N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N $\overleftarrow{prep}$ N	18569
N $\overrightarrow{nsubj}$ N $\overleftarrow{rcmod}$ V $\overleftarrow{prep}$ N	18124
N $\overrightarrow{nsubj}$ N $\overleftarrow{prep}$ N $\overleftarrow{prep}$ N	12316
N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N	11203
N $\overrightarrow{nsubjpass}$ V $\overleftarrow{agent}$ N	9588
N $\overrightarrow{nsubj}$ N $\overleftarrow{partmod}$ V $\overleftarrow{prep}$ N	7480
N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N $\overleftarrow{prep}$ N	7160
N $\overrightarrow{nsubjpass}$ V $\overleftarrow{prep}$ N $\overleftarrow{prep}$ N	5008
N $\overrightarrow{nsubj}$ V $\overleftarrow{conj\_and}$ V $\overleftarrow{prep}$ N	3119
N $\overrightarrow{nsubj}$ N $\overleftarrow{partmod}$ V $\overleftarrow{agent}$ N	2868
N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N $\overleftarrow{conj\_and}$ N	2840
N $\overrightarrow{nsubj}$ V $\overleftarrow{ccomp}$ V $\overleftarrow{prep}$ N	2696
N $\overrightarrow{nsubj}$ V $\overleftarrow{xcomp}$ V $\overleftarrow{prep}$ N	2663
N $\overrightarrow{nsubjpass}$ V $\overleftarrow{conj\_and}$ V $\overleftarrow{prep}$ N	2464
N $\overrightarrow{nsubjpass}$ V $\overleftarrow{dobj}$ N	2317

Table 5.1: Top 20 extraction patterns constructed by WOE using Wikipedia.

where f_{max} (50,259 in this work) is the maximal frequency of pattern in DB_p, and f_{min} (set 1 in this work) is the controlling threshold that determines the minimal frequency of a valid pattern. Note we do not use negative examples when computing weights for patterns in equation 5.1. We suspect that adding a penalty to $w(p)$ based on how often p appears in negative examples might result in a better weighting function. However, it is very difficult to automatically label negative examples for a pattern $p \in DB_p$. Take the sentence "*Tom was born in Boston in 1980; he was sent to Seattle in WA*" for example. If "*Boston*" is matched with Tom's "*birthPlace*" attribute in the infobox, WOE can confidently label $\langle Tom, was\ born\ in, Boston\rangle$ as a positive example for the pattern "$N\ \overrightarrow{nsubjpass}\ V\ \overleftarrow{prep}\ N$"; however, for other tuples of the same pattern, like $\langle Tom, was\ sent\ to, Seattle\rangle$ and $\langle Tom, was\ sent\ in, WA\rangle$, it is hard for WOE to decide whether they are positive or negative examples.

Given a test sentence, like the previous one "*Dan was not born in Berkeley*", WOE^{parse} first identifies *Dan* as $\texttt{arg}_1$ and *Berkeley* as $\texttt{arg}_2$ based on NP-chunking. It then computes the corePath "*Dan* $\overrightarrow{nsubjpass}$ *born* $\overleftarrow{prep\_in}$ *Berkeley*" and abstracts to p="$N\ \overrightarrow{nsubjpass}\ V\ \overleftarrow{prep}\ N$". It then queries DB_p to retrieve the frequency $f_p = 29112$ and assigns a probability of 0.95. Finally, WOE^{parse} traverses the triple's expandPath to output the final expression $\langle Dam, was\ not\ born\ in, Berkeley\rangle$. As shown in the experiments on three corpora, WOE^{parse} achieves an F-measure which is between 51% to 70% greater than TextRunner's.

Extraction with Shallow Features

WOE^{parse} has a dramatic performance improvement over TextRunner. However, the improvement comes at the cost of speed — TextRunner runs about $30X$ faster by eliminating the need for parsing. Since high speed can be crucial when processing Web-scale corpora, we additionally learn a CRF extractor WOE^{pos} based on shallow features like POS-tags. In both cases, however, we generate training data from Wikipedia by matching sentences with infoboxes, while TextRunner used a small set of hand-written rules to label training examples from the Penn Treebank.

We use the same matching sentence set behind DB_p to generate positive examples for WOE^{pos}. Specifically, for each matching sentence, we label the subject and infobox attribute value as $\texttt{arg}_1$ and $\texttt{arg}_2$ to serve as the ends of a linear CRF chain. Tokens involved in the expandPath are labeled as `rel`. Negative examples are generated from random noun-phrase pairs in other sentences when the generalized-CorePaths between the NP pairs are not in DB_p. In total, we generated 170,493 positive and 77,381 negative examples for training WOE^{pos} in our experiment.

WOE^{pos} uses the same learning algorithm and selection of features as TextRunner: a second-order CRF chain model is trained with the Mallet package [74]. WOE^{pos}'s features include POS-tags, regular expressions (*e.g.*, for detecting capitalization, punctuation, *etc.*.), and conjunctions of features occurring in adjacent positions within six words to the left and to the right of the current word.

As shown in the experiments, WOE^{pos} achieves an improved F-measure over TextRunner between 9% to 23% on three corpora, and this is mainly due to the increase on precision.

5.3 Experiments

We created three test datasets[4] by randomly selecting 300 sentences from each of the following corpora: WSJ from Penn Treebank, Wikipedia, and the general Web. Each sentence was examined by two people to label all reasonable triples. These candidate triples are mixed with pseudo-negative ones and submitted to Amazon Mechanical Turk for verification. Each triple was examined by 5 Turkers. We mark a triple's final label as positive when more than 3 Turkers marked them as positive.

5.3.1 Overall Performance Analysis

In this section, we compare the overall performance of WOE^{parse}, WOE^{pos} and TextRunner (shared by the Turing Center at the University of Washington). In particular, we answer the following questions:

[4]Available at: http://ai.cs.washington.edu/www/media/downloadable/media/300Sent-benchmark-WOE.zip

- How do these systems perform against each other?
- How does performance vary w.r.t. sentence length?
- How does extraction speed vary w.r.t. sentence length?

Overall Performance Comparison

The detailed P/R curves are shown in Figure 5.7. To have a close look, for each corpus, we randomly divided the 300 sentences into 5 groups and compared the best F-measures of three systems in Figure 5.6 and Table 5.2. We can see that:

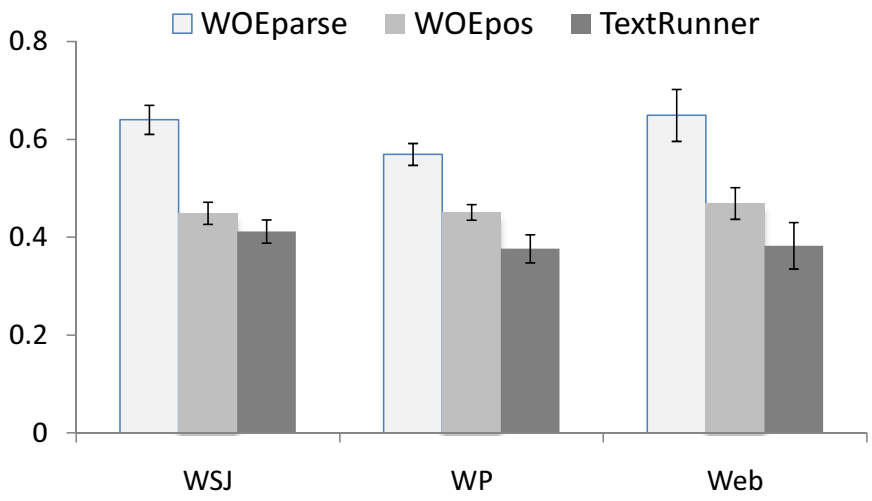


Figure 5.6: WOE^{pos}achieves an F-measure, which is between 9% and 23% better than TextRunner's. WOE^{parse}achieves an improvement between 51% and 70% over TextRunner. The error bar indicates one standard deviation.

- WOE^{pos} is better than TextRunner, especially on precision. This is due to better training data from Wikipedia via self-supervision. Section 5.3.2 discusses this in more detail.
- WOE^{parse} achieves the best performance, especially on recall. This is because the parser features help to handle complicated and long-distance relations in difficult

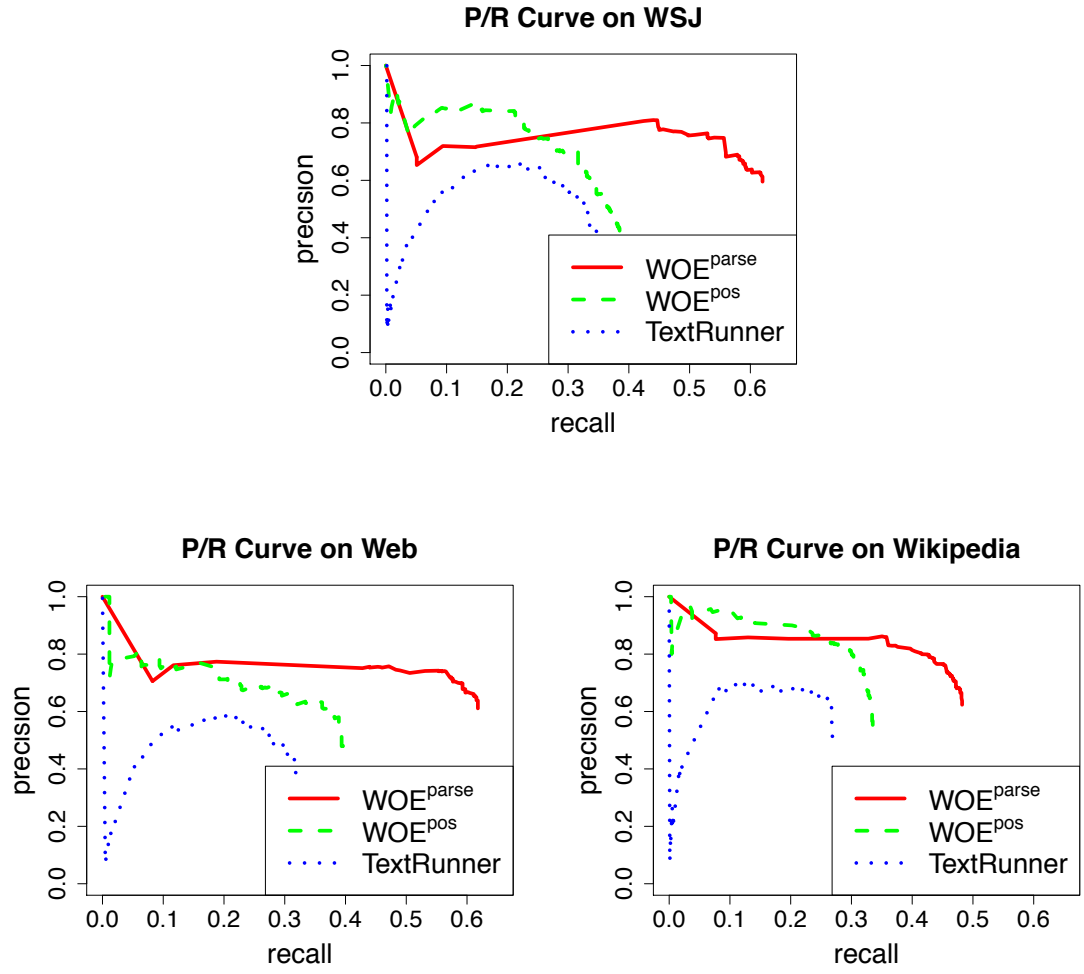


Figure 5.7: WOE^{pos} performs better than TEXTRUNNER, especially on precision. WOE^{parse} dramatically improves performance, especially on recall.

Corpus	Extractor	Precision	Recall	F-measure
WSJ	WOE^{parse}	**0.723**	**0.587**	**0.647**
	WOE^{pos}	0.648	0.352	0.453
	TextRunner	0.583	0.327	0.419
Web	WOE^{parse}	**0.728**	**0.590**	**0.650**
	WOE^{pos}	0.619	0.385	0.471
	TextRunner	0.497	0.309	0.380
WP	WOE^{parse}	0.746	**0.465**	**0.572**
	WOE^{pos}	**0.771**	0.315	0.447
	TextRunner	0.652	0.265	0.376

Table 5.2: Extractors' performance comparison on three corpora.

sentences. In particular, WOE^{parse} outputs 2.1 triples per sentence on average, while WOE^{pos} outputs 1.7 and TextRunner outputs 1.5.

The extraction errors by WOE^{parse} can be categorized into four classes. We illustrate them with the WSJ corpus. In total, WOE^{parse} made 85 incorrect extractions on WSJ, and they were caused by: 1) Incorrect $\texttt{arg}_1$ and/or $\texttt{arg}_2$ from NP-Chunking (18.6%); 2) A erroneous dependency parse from Stanford Parser (11.9%); 3) Inaccurate meaning (27.1%) — for example, $\langle she, isNominatedBy, PresidentBush\rangle$ is wrongly extracted from the sentence "If she is nominated by President Bush ..."[5]; 4) A pattern inapplicable for the test sentence (42.4%).

Recall that WOE automatically labeled 301,962 sentences for training open extractors by matching infobox attribute values with Wikipedia articles. We randomly selected 200 matching sentences and found 167 (83.5%) of them were correctly annotated. The errors can be classified into three categories: 1) the attribute value is mentioned in the sentence, but for a different relation than the target one (13%) — for example, "England" is wrongly

[5]These kind of errors might be excluded by monitoring whether sentences contain words such as 'if,' 'suspect,' 'doubt,' *etc.*. We leave this as a topic for the future.

Corpus	Pattern	Num of Extractions	Precision	Pattern Rank in DB_p
WSJ	N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N	200	0.86	8
	N $\overrightarrow{nsubj}$ N	48	0.71	4
	N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N	47	0.68	1
	N $\overrightarrow{nsubjpass}$ V $\overleftarrow{prep}$ N	33	0.82	3
	N $\overrightarrow{nsubj}$ V $\overleftarrow{ccomp}$ V $\overleftarrow{dobj}$ N	32	0.03	85
	N $\overleftarrow{partmod}$ V $\overleftarrow{prep}$ N	18	0.61	22
	N $\overrightarrow{nsubj}$ V $\overleftarrow{xcomp}$ V $\overleftarrow{dobj}$ N	15	0.47	31
	N $\overrightarrow{nsubj}$ V $\overleftarrow{conj\_and}$ V $\overleftarrow{dobj}$ N	14	0.93	40
	N $\overleftarrow{partmod}$ V $\overleftarrow{agent}$ N	13	0.77	69
	N $\overrightarrow{nsubj}$ V $\overleftarrow{ccomp}$ V $\overleftarrow{prep}$ N	12	0.08	16
WP	N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N	151	0.85	8
	N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N	86	0.87	1
	N $\overrightarrow{nsubj}$ N	77	0.84	4
	N $\overrightarrow{nsubjpass}$ V $\overleftarrow{prep}$ N	60	0.87	3
	N $\overleftarrow{partmod}$ V $\overleftarrow{prep}$ N	29	0.72	22
	N $\overrightarrow{nsubjpass}$ V $\overleftarrow{agent}$ N	22	1.00	9
	N $\overrightarrow{nsubj}$ V $\overleftarrow{xcomp}$ V $\overleftarrow{dobj}$ N	14	0.64	31
	N $\overrightarrow{nsubj}$ V $\overleftarrow{conj\_and}$ V $\overleftarrow{dobj}$ N	13	0.77	40
	N $\overleftarrow{partmod}$ V $\overleftarrow{agent}$ N	12	0.75	69
	N $\overleftarrow{partmod}$ V $\overleftarrow{dobj}$ N	11	0.36	67
Web	N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N	143	0.73	8
	N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N	51	0.71	1
	N $\overrightarrow{nsubj}$ N	39	0.79	4
	N $\overrightarrow{nsubjpass}$ V $\overleftarrow{prep}$ N	16	0.94	3
	N $\overrightarrow{nsubj}$ V $\overleftarrow{conj\_and}$ V $\overleftarrow{dobj}$ N	13	0.77	40
	N $\overrightarrow{nsubj}$ V $\overleftarrow{dobj}$ N $\overleftarrow{conj\_and}$ N	12	0.92	38
	N $\overrightarrow{nsubj}$ V $\overleftarrow{xcomp}$ V $\overleftarrow{dobj}$ N	11	0.64	31
	N $\overleftarrow{partmod}$ V $\overleftarrow{prep}$ N	11	0.45	22
	N $\overrightarrow{nsubj}$ V $\overleftarrow{ccomp}$ V $\overleftarrow{dobj}$ N	11	0.18	85
	N $\overrightarrow{nsubj}$ V $\overleftarrow{prep}$ N $\overleftarrow{conj\_and}$ N	7	0.86	15

Table 5.3: Top 10 patterns that have the most extractions for each corpus, showing that patterns with higher frequencies tend to have more extractions with higher precision.

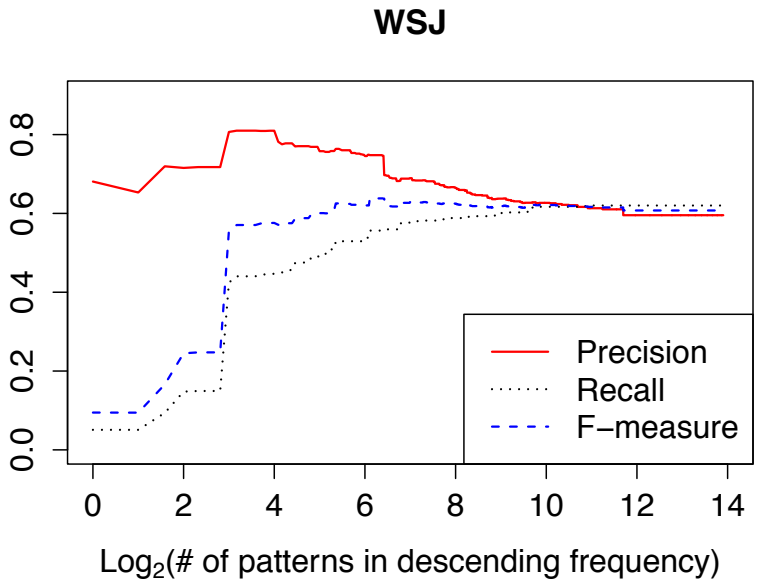


Figure 5.8: Although the top frequent patterns capture most extractions, the less frequent ones are helpful to further increase the recall.

annotated as the "birthplace" for "Greg Lincoln" in the sentence "He represented England at Under-20 level."[6]; 2) the infobox record is wrong (1.5%) — for example, "1936" is wrongly set as the "birthplace" attribute in the infobox for "Doug" by an editor, which in turn leads to the wrong matching sentence "Doug Kershaw, born January 24, 1936, ..."; 3) $\texttt{arg}_1$ is wrongly annotated (2%) — for example, "the band" is incorrectly labeled as $\texttt{arg}_1$ of the "starring" relation for the sentence "It starred the band: Singer Tony...". If we are going to learn relation-specific extractors, these wrongly matched sentences will definitely hurt the extractors' performance. However, for learning relation-independent open extractors, some of these errors happen to be fine — especially those from the first two categories. Take the previous sample sentence "He represented England at Under-20 level." for example. Although "England" is not the "birthplace" of "Greg Lincoln" in

[6]Sometimes this kind of error is tricky to classify. For example, "Eagle" is marked as the "currentTeam" for "Tom" in the sentence "Tom was traded to Eagle in 1998."; however, it is unclear whether he is still playing there. We took a strict policy which considers such cases to be wrong matches.

this sentence , $\langle He, represented, England\rangle$ is still a prefect instance for inducing general extraction patterns for open extractors.

As described in Section 5.2.3, WOE builds a database DB_p with 15,333 distinct patterns based on those 301,962 matching sentences. To have a closer look at these patterns' effects on distilling facts from sentences, we gradually increased the number of allowed patterns in the decreasing order of frequency, and computed the precison/recall/f-measure at each point. The result on the WSJ corpus[7] is in Figure 5.8, showing that although the top frequent patterns capture most extractions, the less frequent ones are helpful to further increase the recall. Table 5.3 shows the patterns that have the most extractions for each corpus. We can see that patterns with higher frequencies tend to have more extractions with higher precision.

We also tested the effect of relaxing some sentence matching heuristics when building the pattern database DB_p. Although this will increase the number of patterns in DB_p (e.g., from 15,333 patterns to 29,858 patterns after allowing multiple sentences to match one infobox attribute value and imposing no requirement for `arg`s to be heads of NP-chunks), it has little effect on WOE's extraction performance. Most likely, this is because most additional patterns via relaxing the sentence matching heuristics sit on the long tail of the pattern distribution. There is little change to the distribution of the top few thousand patterns on the head, which largely determine WOE's performance.

Note WOE^{parse} is worse than WOE^{pos} in the low recall region. This is mainly due to parsing errors (especially on long-distance dependencies), which misleads WOE^{parse} to extract false high-confidence triples. WOE^{pos} won't suffer from such parsing errors. Therefore it has better precision on high-confidence extractions.

We noticed that TextRunner has a dip point in the low recall region. There are two typical errors responsible for this. A sample error of the first type is $\langle Source, sold, the\ company\rangle$ extracted from the sentence "Sources said he sold the company 2 weeks ago", where "Sources" is wrongly treated as the subject of the object clause. A sample error of the second type is $\langle this\ year, will\ star\ in, the\ new\ movie\rangle$ extracted from the sentence "Coming up this year,

[7]Results on the Web and Wikipedia corpora are similar.

Extractor	Precision	Recall	F-measure
WOE^{parse}	**0.920**	**0.398**	**0.555**
WOE^{pos}	0.882	0.394	0.544
TextRunner	0.879	0.397	0.547

Table 5.4: All extractors are comparable to each other, and achieve hight precisions and decent recalls on the 500SENT set when only counting the concrete triples that contain preset arguments of certain target relations.

Long will star in the new movie.", where "this year" is wrongly treated as part of a compound subject. Taking the WSJ corpus for example, at the dip point with recall=0.002 and precision=0.08, these two types of errors account for 70% of all errors.

Note that we measure TextRunner's precision & recall differently than in [14, 16]. Specifically, we compute the precision & recall based on *all* extractions, while Banko *et al.* counted only *concrete* triples where $\texttt{arg}_1$ is a proper noun and $\texttt{arg}_2$ is a proper noun or date (they also required the frequency of `rel` to be over a threshold in many cases). To have a closer comparison with TextRunner on *concrete* extractions, we also tested extractors' performance on the 500SENT set from [16]. Each sentence in this dataset describes one of the four target relations (*acquisition, invention, hometown*, and *award*) and a pair of proper nouns ($\texttt{arg}_1$ and $\texttt{arg}_2$ of the relation) are marked. We ran each extractor on this dataset and only counted the extractions which have the marked proper noun pairs as $\texttt{arg}_1$ and $\texttt{arg}_2$. We labeled the `rel` by ourselves given they are not provided in the original 500SENT dataset. The comparisons of three extractors are in Table 5.4, showing that: 1) the three extractors are comparable to each other on this dataset; 2) when $\texttt{arg}_1$ and $\texttt{arg}_2$ are preset, all extractors achieve high precisions and decent recalls. Note again that unlike the previous experiments where we measure *true* precision & recall based on *all* extractions, the numbers here are measured only based on those concrete triples that contain two preset arguments of certain target relations. We note that the performance of TextRunner is slightly different than that reported in [16], mostly likely due to some difference in labeling `rel` in sentences.

Extraction Performance *vs.* **Sentence Length**

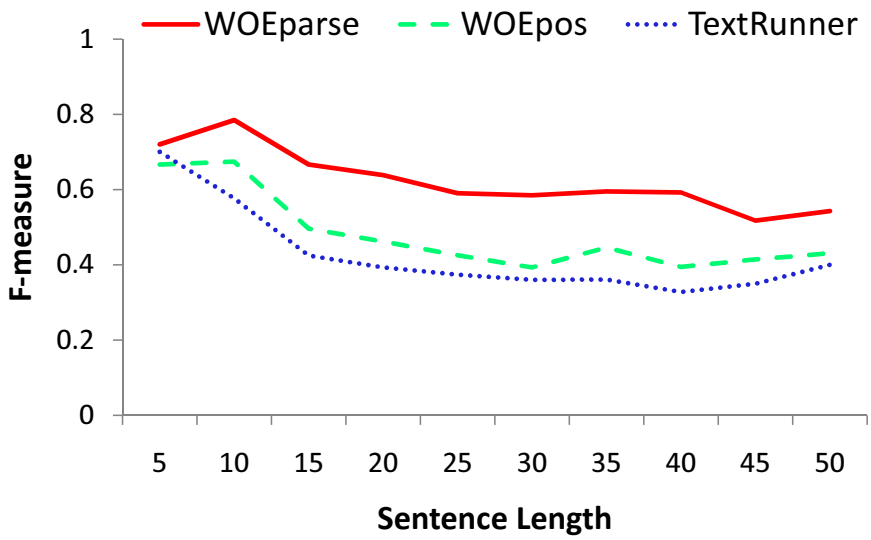


Figure 5.9: WOE^{parse}'s F-measure decreases more slowly with sentence length than WOE^{pos} and TEXTRUNNER, due to its better handling of difficult sentences using parser features.

We tested how extractors' performance varies with sentence length; the results are shown in Figure 5.9. TEXTRUNNER and WOE^{pos} have good performance on short sentences, but their performance deteriorates quickly as sentences get longer. This is because long sentences tend to have complicated and long-distance relations which are difficult for shallow features to capture. In contrast, WOE^{parse}'s performance decreases more slowly w.r.t. sentence length. This is mainly because parser features are more useful for handling difficult sentences and they help WOE^{parse} to maintain a good recall with only moderate loss of precision.

Extraction Speed *vs.* **Sentence Length**

We also tested the extraction speed of different extractors. We used Java for implementing the extractors, and tested on a Linux platform with a 2.4GHz CPU and 4G memory. On average, it takes WOE^{parse} 0.679 seconds to process a sentence. For TEXTRUNNER and WOE^{pos}, it only takes 0.022 seconds — 30X times faster. The detailed extraction speed *vs.* sentence length is in Figure 5.10, showing that TEXTRUNNER and WOE^{pos}'s extraction time grows approximately linearly with sentence length, while WOE^{parse}'s extraction time

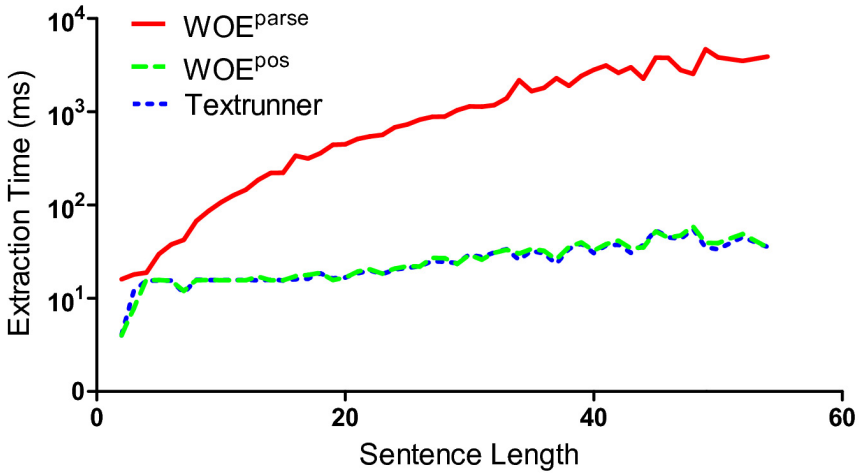


Figure 5.10: TEXTRUNNER and WOE^{pos}'s running time seems to grow linearly with sentence length, while WOE^{parse}'s time grows quadratically.

grows quadratically ($R^2 = 0.935$) due to its reliance on parsing.

5.3.2 Self-Supervision with Wikipedia Results in Better Training Data

In this section, we consider how the process of matching Wikipedia infobox values to corresponding sentences results in better training data than the hand-written rules used by TEXTRUNNER.

To compare with TEXTRUNNER, we tested four different ways to generate training examples from Wikipedia for learning a CRF extractor. Specifically, positive and/or negative examples are selected by TEXTRUNNER's hand-written rules (*tr* for short), by WOE's heuristic of matching sentences with infoboxes (w for short), or randomly (r for short). We use CRF$_{+h_1-h_2}$ to denote a particular approach, where "+" means positive samples, "-" means negative samples, and $h_i \in \{tr, w, r\}$.

In particular, "$+w$" results in 170,493 positive examples based on the matching sentence set[8] and "$-w$" results in 77,381 negative examples. All extractors are trained using about the same number of positive and negative examples. In contrast, TEXTRUNNER was trained

[8]This number is smaller than the total number of corePaths (259,046) because we require $\texttt{arg}_1$ to appear before $\texttt{arg}_2$ in a sentence — as specified by TEXTRUNNER.

with 91,687 positive examples and 96,795 negative examples generated from the WSJ dataset in Penn Treebank. The CRF extractors are trained using the same learning algorithm and feature selection as TextRunner. The detailed P/R curves are in Figure 5.11, showing that using WOE heuristics to label positive examples gives the biggest performance boost. CRF$_{+tr-tr}$ (trained using TextRunner's heuristics) is slightly worse than TextRunner. Most likely, this is because TextRunner's heuristics rely on parse trees to label training examples, and the Stanford parse on Wikipedia is less accurate than the gold parse on WSJ.

We redid the comparison experiment by enforcing all extractors to be trained using the exact same number of positive (91,687) and negative (96,795) examples as TextRunner. The new P/R curves are in Figure 5.12, showing the same conclusions as before. WOE^{pos} has a slightly worse performance due to less training examples.

5.3.3 Design Desiderata of WOE$^{\text{parse}}$

There are two interesting design choices in WOE^{parse}:

- whether to require $\texttt{arg}_1$ to appear before $\texttt{arg}_2$ (denoted as $1 \prec 2$) in the sentence;
- whether to allow corePaths to contain prepositional phrase (PP) attachments (denoted as *PPa*);

We tested how these choices affect the extraction performance; the results are shown in Figure 5.13. We can see that filtering PP attachments ($\overline{PPa}$) gives a large precision boost with a noticeable loss in recall; enforcing a lexical ordering of relation arguments ($1 \prec 2$) yields a smaller improvement in precision with small loss in recall. Take the WSJ corpus for example: setting $1 \prec 2$ and $\overline{PPa}$ achieves a precision of 0.748 (with recall of 0.556). By changing $1 \prec 2$ to $1 \sim 2$, the precision decreases to 0.730 (with recall of 0.591). By changing $\overline{PPa}$ to *PPa* and keeping $1 \prec 2$, the precision decreases to 0.580 (with recall of 0.662) — in particular, if we use gold parse, the precision decreases to 0.613 (with recall of 0.664). We set $1 \prec 2$ and $\overline{PPa}$ as default in WOE^{parse} as a logical consequence of our preference for high precision over high recall.

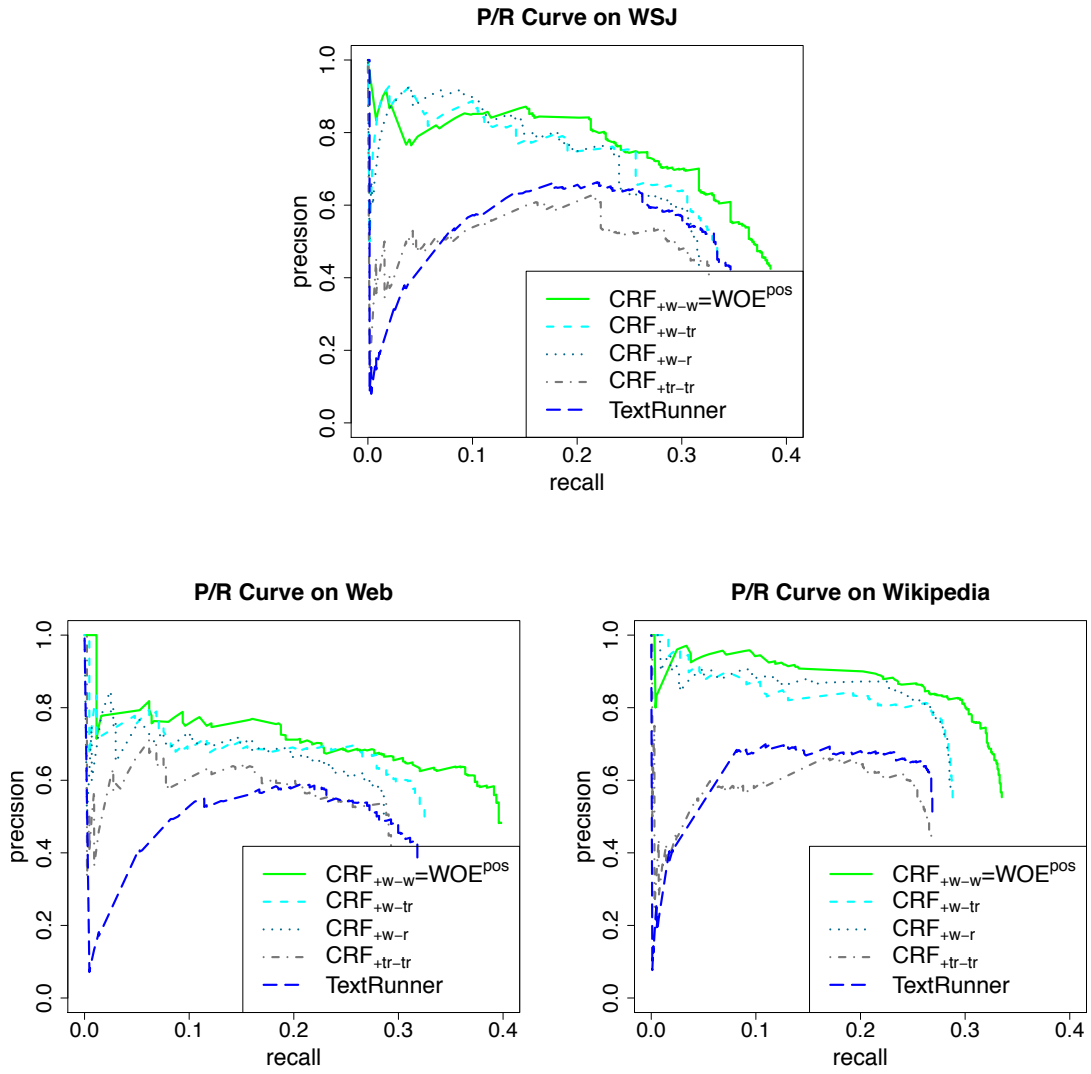


Figure 5.11: Matching sentences with Wikipedia infoboxes results in better training data than the hand-written rules used by TextRunner.

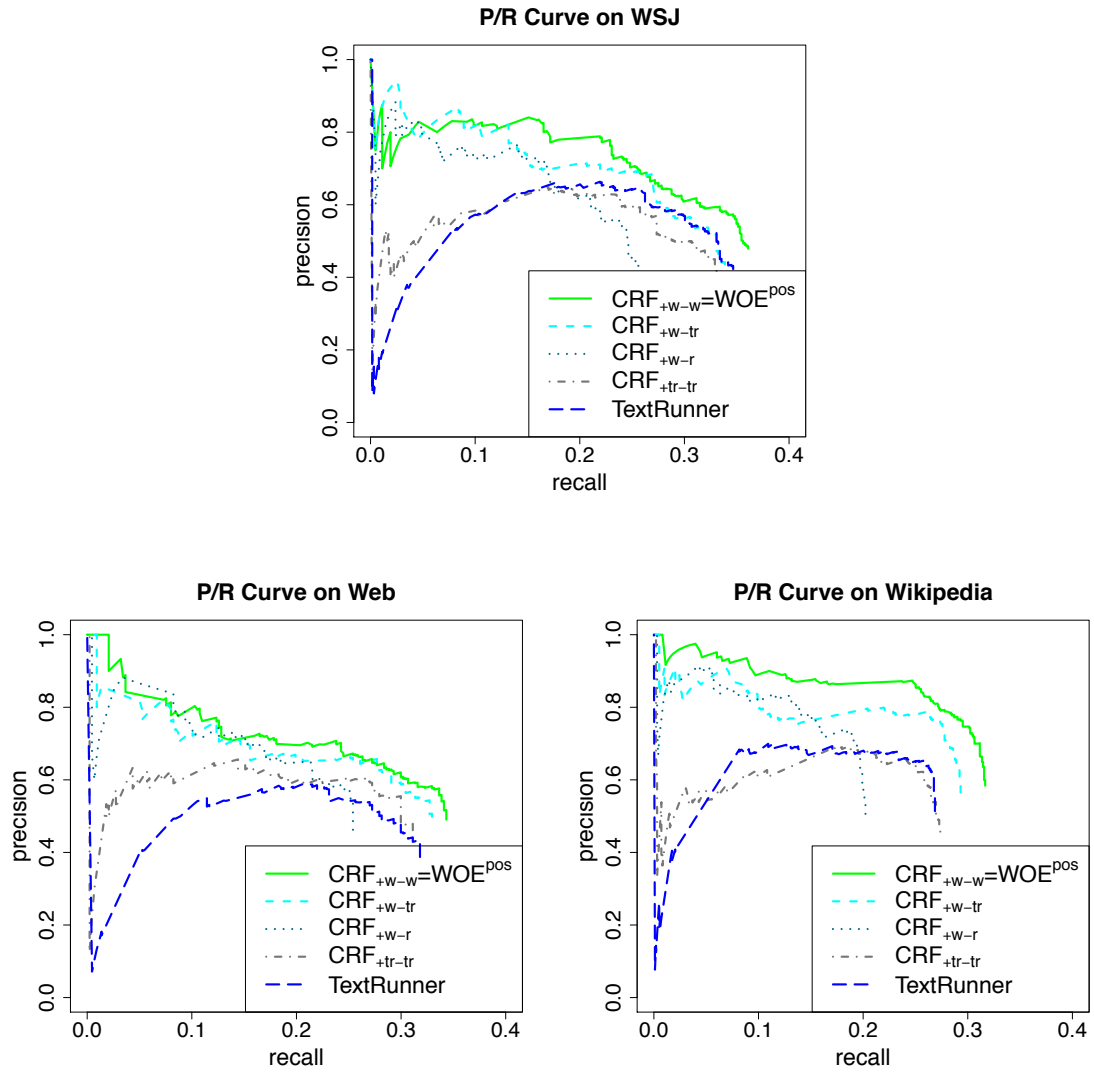


Figure 5.12: Although WOE^{pos}'s performance decreases a bit due to less training examples than in Figure 5.11, it still shows that matching sentences with Wikipedia infoboxes results in better training data than the hand-written rules used by TextRunner.

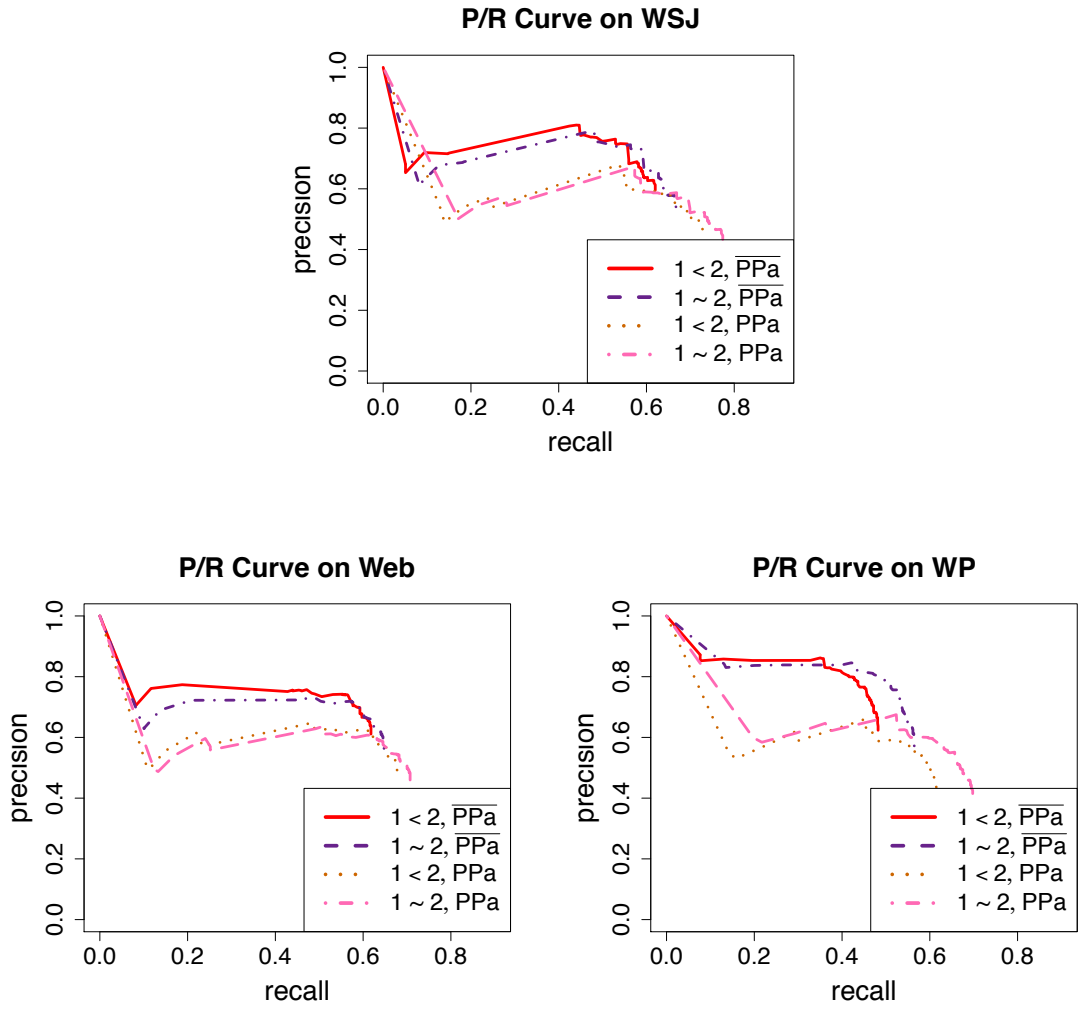


Figure 5.13: Filtering prepositional phrase attachments ($\overline{PPa}$) shows a strong boost to precision, and we see a smaller boost from enforcing a lexical ordering of relation arguments ($1 \prec 2$).

5.3.4 Different Parsing Options

We also tested how different parsing might effect WOE^{parse}'s performance. We used three parsing options on the WSJ dataset: Stanford parsing, CJ50 parsing [30], and the gold parses from the Penn Treebank. The Stanford Parser is used to derive dependencies from CJ50 and gold parse trees. Figure 5.14 shows the detailed P/R curves. We can see that although today's statistical parsers make errors, they have negligible effect on the accuracy of WOE.

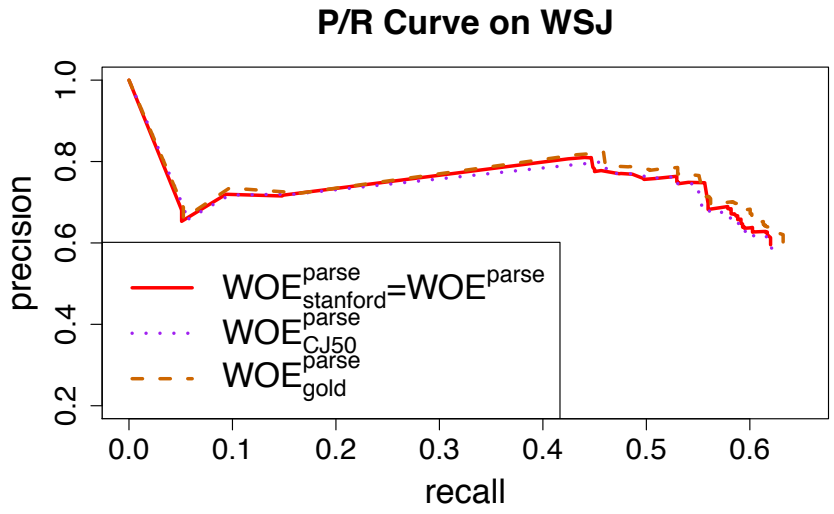


Figure 5.14: Although today's statistical parsers make errors, they have negligible effect on the accuracy of WOE compared to operation on gold standard, human-annotated data.

5.3.5 Trade Recall for Precision via Filters

As mentioned in Section 5.3.1 that Banko *et al.* applied two filters when computing the precision & recall for TextRunner [14]:

- **Concreteness:** a concrete tuple, $\langle \texttt{arg}_1, \texttt{rel}, \texttt{arg}_2 \rangle$, has $\texttt{arg}_1$ as a proper noun and $\texttt{arg}_2$ as a proper noun or date. Regular expressions and POS tags are used to identify concrete tuples.

- **Distributional constraints:** the following distributional constraints are imposed on the entire set of tuples: we discard tuples in which `rel` occurs with fewer than n distinct facts in the entire extraction set; `rel` must be observed with at least e_1 unique $\texttt{arg}_1$ and e_2 unique $\texttt{arg}_2$.

One can image applying these filters to trade recall for precision for any open extractor. We tested their effects on the three corpora. Specifically, we ran TextRunner, WOE^{parse}, and WOE^{pos}over the whole Wikipedia corpus for computing the distributional constraints. We set $n = 50$, $e_1 = 50$, and $e_2 = 20$ following the recommendations in [13]. The detailed experimental results are in Table 5.5 when setting 0.5 as the confidence threshold. We can see that the "distribution" filter improves precision with modest loss of recall in most cases; the "concreteness" filter improves precision with big loss of recall on the Wikipedia and Web corpora, but hurts all extractors' precisions on the WSJ corpus; when combined together, these two filters improve precision in all cases except for TextRunner on the WSJ corpus.

Given "distribution" filter's promising performance of improving precision with modest loss of recall, we further tested how different distributional constraints' thresholds might affect extractors' performance. Specifically, we set two of the three thresholds as 0 (e.g., $e_1 = 0$, and $e_2 = 0$), varied the third threshold from 0 to 50 (e.g., $n = 0, ..., 50$), and computed the precision and recall at each point. Figure 5.15 shows the results for WOE^{parse} on the WSJ corpus. We can see that there is a jump in precision when increasing the distributional constraint threshold from 0 to a small number for n, e_1, or e_2; afterwards, the curves become very flat. Also, the results are very similar for n, e_1, and e_2 because they are tightly co-related — if a `rel` has a large number of unique facts, it also tends to have a large number of unique $\texttt{arg}_1$ and $\texttt{arg}_2$.

5.3.6 Open vs. Traditional IE

We are also interested in comparing open extractors with traditional ones (like our KYLIN described in Chapter 2). However, since open IE extracts arbitrary relations from sentences while traditional IE only outputs target relations, designing the comparison metric is subjective. In our case, we created a test dataset by sampling 50 Wikipedia sentences for five

Corpus	Extractor	Filters	# of Tuples	Pre.	Rec.	F-mea.
WSJ	WOE^{parse}	None	510	0.689	**0.56**	**0.618**
		Distribution	337	0.82	0.44	0.573
		Concrete	15	0.6	0.014	0.028
		Concrete+Distribution	7	**0.857**	0.01	0.019
	WOE^{pos}	None	453	0.507	**0.366**	**0.425**
		Distribution	352	0.518	0.291	0.373
		Concrete	11	0.455	0.008	0.016
		Concrete+Distribution	4	**0.75**	0.005	0.009
	TextRunner	None	396	0.521	**0.329**	**0.404**
		Distribution	293	**0.567**	0.264	0.36
		Concrete	8	0.5	0.006	0.013
		Concrete+Distribution	4	0.5	0.003	0.006
Web	WOE^{parse}	None	352	0.713	**0.574**	**0.636**
		Distribution	247	0.745	0.421	0.538
		Concrete	3	0.667	0.005	0.009
		Concrete+Distribution	0	**1.0**	0.0	0.0
	WOE^{pos}	None	287	0.585	**0.384**	**0.464**
		Distribution	212	0.571	0.277	0.373
		Concrete	3	0.667	0.005	0.009
		Concrete+Distribution	2	**1.0**	0.005	0.009
	TextRunner	None	285	0.46	**0.3**	**0.363**
		Distribution	202	0.505	0.233	0.319
		Concrete	3	0.667	0.005	0.009
		Concrete+Distribution	2	**1.0**	0.005	0.009
WP	WOE^{parse}	None	607	0.736	**0.458**	**0.564**
		Distribution	482	0.829	0.41	0.548
		Concrete	45	0.756	0.035	0.066
		Concrete+Distribution	28	**0.893**	0.026	0.05
	WOE^{pos}	None	467	0.678	**0.325**	**0.439**
		Distribution	400	0.699	0.287	0.407
		Concrete	37	0.649	0.025	0.047
		Concrete+Distribution	26	**0.808**	0.021	0.042
	TextRunner	None	413	0.624	**0.265**	**0.372**
		Distribution	332	0.694	0.236	0.352
		Concrete	29	0.759	0.022	0.044
		Concrete+Distribution	22	**0.818**	0.018	0.036

Table 5.5: The "distribution" filter improves precision with modest loss of recall in most cases; the "concreteness" filter improves precision with big loss of recall on the Wikipedia and Web corpora, however, it hurts all extractors' precisions on the WSJ corpus; when combined together, they improve precision in all cases except for TextRunner on the WSJ corpus.

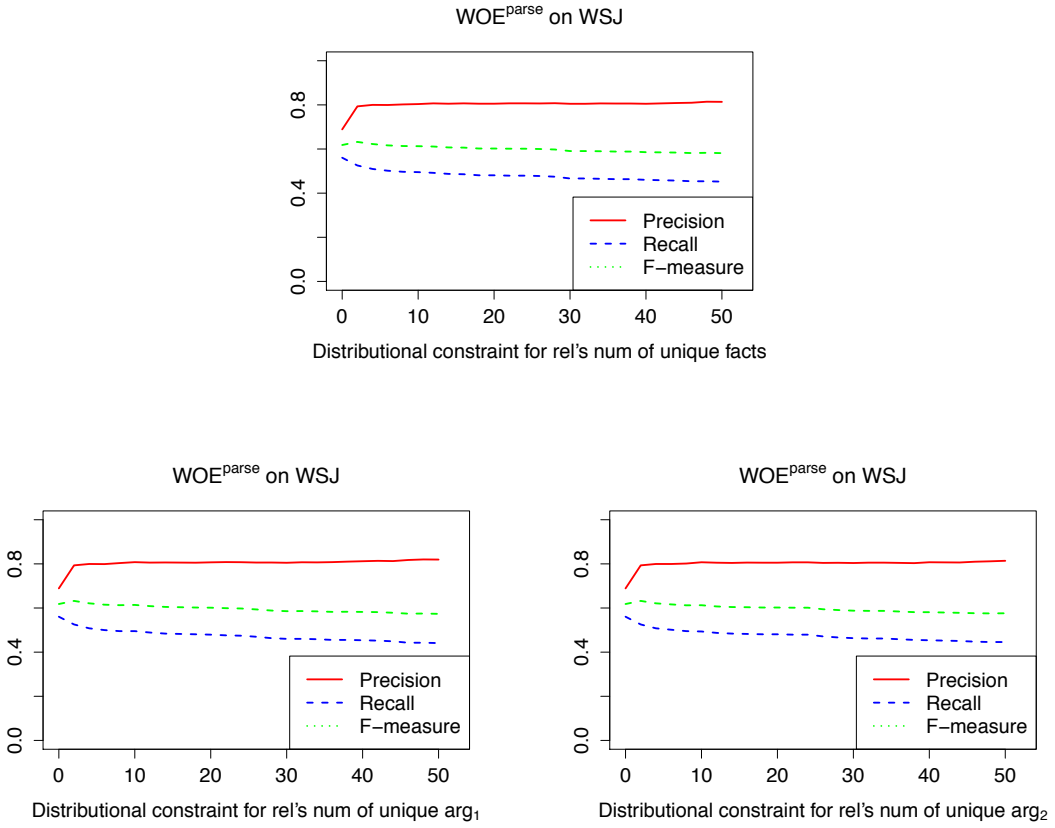


Figure 5.15: There is a jump in precision when increasing the distributional constraint from 0 to a small threshold for n, e_1, or e_2; afterwards, the curves become very flat.

relations (*person.birthPlace, person.spouse, university.establishment, university.nickname,* and *county.water_square_mile*) — 10 sentences per relation[9]. We ran KYLIN, WOE^{parse}, WOE^{pos}, and TEXTRUNNER over the sentences and computed F-measures only based on the tuples related to the target relations. Specifically, all tuples from KYLIN were counted; for open extractors, a tuple $\langle \texttt{arg}_1, \texttt{rel}, \texttt{arg}_2 \rangle$ is counted **iff** `rel` expresses a target relation (e.g., "was married to" for *person.spouse*), or $\texttt{arg}_2$ matches the value for the target relation (e.g., "Boston" for *person.birthPlace* in the sentence "Dan was born in Bonston"). Table 5.6 shows the detailed performance of each extractor on this dataset. Based on this preliminary

[9]Available at http://ai.cs.washington.edu/www/media/downloadable/media/50SENT.zip

experiment, we can see that both WOE^{pos} and TextRunner perform worse than KYLIN on this 5-relation test dataset; WOE^{parse} has higher recall but lower precision than KYLIN and achieves a better F-measure. We note this result is a little different than that in [16] where the reported precisions of traditional extractors were lower (about 70%). Most likely, there are two reasons for this: 1) we labeled ground-truth tuples differently than Banko *et al.* did in [16]; 2) our test dataset is constructed using Wikipedia and all sentences are positive examples, which help KYLIN to achieve high precisions. In the future, we plan to do more experiments to explore the relationships between open and traditional IE and study how to combine them together for a hybrid and better extractor.

5.4 Related Work

Open or Traditional Information Extraction: Most existing work on IE is relation-specific. Occurrence-statistical models [8, 69], graphical models [86, 90], and kernel-based methods [25] have been studied. Snow *et al.* [101] utilize WordNet to learn dependency path patterns for extracting the hypernym relation from text. Some seed-based frameworks are proposed for open-domain extraction [39, 40, 84]. These works focus on identifying general relations such as class attributes, while open IE aims to extract relation instances from given sentences. Another seed-based system StatSnowball [114] can perform both relation-specific and open IE by iteratively generating weighted extraction patterns. Different from WOE, StatSnowball only employs shallow features and uses L1-normalization to weight patterns. Shinyama and Sekine proposed the "preemptive IE" framework to avoid relation-specificity [98]. They first group documents based on pairwise vector-space clustering, then apply an additional clustering to group entities based on documents clusters. The two clustering steps make it difficult to meet the scalability requirement necessary to process the Web. Bollegala *et al.* proposed a two-step procedure for relation extraction [22]. They first generate entity pairs and shallow lexical-syntactic patterns for the pairs from a given text corpus. Then a sequential co-clustering is performed to find clusters for entity pairs and lexical-syntactic patterns iteratively. In contrast, WOE uses generalized parse features to induce patterns for open extraction. Alan Akbik *et al.* [10] annotated 10,000 sentences parsed with LinkGrammar and selected 46 general linkpaths as patterns for relation extrac-

Relation	Extractor	Precision	Recall	F-measure
person.birthPlace	WOE^{parse}	0.667	0.4	0.5
	WOE^{pos}	**1.0**	0.3	0.462
	TextRunner	0.75	0.3	0.429
	KYLIN	**1.0**	**0.5**	**0.667**
person.spouse	WOE^{parse}	**0.889**	**0.8**	**0.842**
	WOE^{pos}	0.625	0.5	0.556
	TextRunner	0.714	0.5	0.588
	KYLIN	0.8	0.4	0.533
university.establishment	WOE^{parse}	**1.0**	**0.7**	**0.824**
	WOE^{pos}	**1.0**	0.0	0.0
	TextRunner	0.5	0.1	0.167
	KYLIN	**1.0**	0.4	0.571
university.nickname	WOE^{parse}	0.857	**0.6**	**0.706**
	WOE^{pos}	0.833	0.5	0.625
	TextRunner	**1.0**	0.3	0.462
	KYLIN	**1.0**	0.3	0.462
county.water_sq_mi	WOE^{parse}	0.667	0.6	0.632
	WOE^{pos}	0.214	0.3	0.25
	TextRunner	0.273	0.3	0.286
	KYLIN	**1.0**	**0.8**	**0.889**
Average	WOE^{parse}	0.816	**0.62**	**0.701**
	WOE^{pos}	0.735	0.32	0.378
	TextRunner	0.647	0.3	0.386
	KYLIN	**0.96**	0.48	0.624

Table 5.6: On a 5-relation test set built using Wikipedia, both WOE^{pos} and TextRunner perform worse than KYLIN; WOE^{parse} has higher recall and lower precision than KYLIN, and achieves a better F-measure than KYLIN.

tion. In contrast, WOE learns 15,333 general patterns based on an automatically annotated set of 301,962 Wikipedia sentences. The KNext system [50] performs open knowledge extraction via significant heuristics. Its output is knowledge represented as logical statements instead of information represented as segmented text fragments.

Unsupervised and Self-Supervised Information Extraction: Some IE systems learn relation-specific extractors *without direct supervision*, *i.e.* without labeled training examples. Our KYLIN system (Chapter 2) is one such example, which automatically creates training dataset via self-supervision using Wikipedia infoboxes, and learns extractors for a broad set of relations. For more discussion on such types of unsupervised or self-supervised IE systems, please refer to Section 2.4 (Related Work).

Shallow or Deep Parsing: Shallow features, like POS tags, enable fast extraction over large-scale corpora [40, 14]. Deep features are derived from parse trees with the hope of training better extractors [26, 112, 113, 106]. Jiang and Zhai [63] did a systematic exploration of the feature space for relation extraction on the ACE corpus. Their results showed limited advantage of parser features over shallow features for IE. However, our results imply that abstracted dependency path features are highly informative for open IE. There might be several reasons for the different observations. First, Jiang and Zhai's results are tested for traditional IE where local lexicalized tokens might contain sufficient information to trigger a correct classification. The situation is different when features are completely unlexicalized in open IE. Second, as they noted, many relations defined in the ACE corpus are short-range relations which are easier for shallow features to capture. In practical corpora like the general Web, many sentences contain complicated long-distance relations. As we have shown experimentally, parser features are more powerful in handling such cases.

5.5 Conclusion

This chapter introduces WOE, a new approach to open IE that uses self-supervised learning over unlexicalized features, based on a heuristic match between Wikipedia infoboxes and corresponding text. WOE can run in two modes: a CRF extractor (WOE^{pos}) trained with

shallow features like POS tags; a pattern classfier (WOE^{parse}) learned from dependency path patterns. WOE^{pos}runs at the same speed as TEXTRUNNER, and achieves an F-measure between 0.447 and 0.471 (*i.e.*, between 9% and 23% greater than that of TEXTRUNNER) on three corpora; WOE^{parse} achieves an F-measure between 0.572 and 0.650 (*i.e.*, between 51% and 70% higher than that of TEXTRUNNER), but runs about $30X$ times slower due to the time required for parsing.

Our experiments uncovered two sources of WOE's strong performance: 1) the Wikipedia heuristic is responsible for the bulk of WOE's improved accuracy, but 2) dependency-parse features are highly informative when performing unlexicalized extraction. We note that this second conclusion disagrees with the findings in [63].

Chapter 6

CONCLUSIONS AND FUTURE WORK

This dissertation has described the challenges associated with machine reading, which we summarize in Section 6.1 below. We argue bootstrapping from Wikipedia is the best way to solve this problem and presented three working systems: KYLIN, KOG, and WOE. These projects make substantial steps toward addressing the challenges of machine reading. They also point to interesting future work involving extensions to each system as well as new overall approaches, as we discuss in Section 6.2.

6.1 Contributions

The vast majority of information is embedded in natural-language texts on the Web. The goal of machine reading is to automatically distill the information from texts and make them accessible to software agents. In Chapter 1 we discussed four design desiderata for an ideal machine reading system. We now briefly examine them before discussing the contributions made by three different systems that embody these criteria.

First, the system should extract information with high accuracy so that subsequent systems can build high-performance applications on top of that. Second, the Web is huge and heterogeneous; there are billions of informative Web documents and they can be arbitrary domains, genres, and languages. Therefore the system should perform large-scale knowledge acquisition from the Web. Third, since the scale of available knowledge is vast, the system should achieve maximal autonomy. Finally, in order to create a comprehensive semantic knowledge base summarizing topics on the Web, the system should conquer both head and tail textural knowledge.

These desiderata motivated the development of our systems. We discuss their contributions in the following sections.

6.1.1 KYLIN Contributions

KYLIN is a relation-specific information extraction system trained using Wikipedia. Since KYLIN uses self-supervised learning, which is bootstrapped on existing user-contributed data, it requires little or no human guidance. We make the following contributions:

- We propose bootstrapping the Semantic Web by mining Wikipedia and we identify some unique challenges (lack of redundancy) and opportunities (unique identifiers, user-supplied training data, lists, categories, etc.) of this approach. We also identify additional issues resulting from Wikipedia's growth through decentralized authoring (e.g., inconsistency, schema drift, etc.). This high-level analysis should benefit future work on Wikipedia and similar collaborative knowledge repositories.

- We describe a system for automatically generating attribute/value pairs summarizing an article's properties. Based on self-supervised learning, KYLIN achieves performance which is roughly comparable with that of human editors. In one case, KYLIN does even better.

- Collaboratively authored data is rife with noise and incompleteness. We identify robust learning methods which can cope in this environment. Extensive experiments demonstrate the performance of our system and characterize some of the crucial architectural choices (e.g., the optimal ordering of heuristics, the utility of classifier-based training data refinement, a pipelined architecture for attribute extraction).

- By applying shrinkage over an automatically-learned subsumption taxonomy, we allow KYLIN to substantially improve the recall of its extractors for sparse infobox classes.

- By mapping the contents of known Wikipedia infobox data to TextRunner, a state-of-the-art open information extraction system [15], we enable KYLIN to clean and augment its training dataset. When applied in conjunction with shrinkage, this *re-training* technique improves recall by a factor of between 1.4 and 5.9, depending on class. It also helps KYLIN to better adapt to the Web corpus and enables KYLIN to

retrieve relevant sentences from the greater Web when it is unable to extract necessary information from a Wikipedia page.

6.1.2 KOG Contributions

KOG automatically generates the Wikipedia Infobox Ontology by combining evidence from heterogeneous resources via joint inference. It embodies several contributions:

- We address the problem of ontology generation and identify the aspects of the Wikipedia data source which facilitate (as well as those which hinder) the refinement process. We codify a set of heuristics which allow these properties to be converted into features for input to machine learning algorithms.

- We cast the problem of subsumption detection as a machine learning problem and solve it using both support-vector machines (SVMs) and Markov Logic Networks (MLNs). The MLNs model is especially novel, simultaneously constructing a subsumption lattice and a mapping to WordNet using joint inference. Our experiments demonstrate the superiority of the joint inference approach and evaluate other aspects of our system.

- Using these techniques, we build a rich ontology which integrates and extends the information provided by both Wikipedia and WordNet; it incorporates both subsumption information, an integrated set of attributes, and attribute mappings between parent and child classes in the subsumption hierarchy.

- We demonstrate how the resulting ontology may be used to enhance Wikipedia in many ways, such as advanced query processing for Wikipedia facts, facetted browsing, automated infobox edits and template generation. Furthermore, we believe that the ontology can benefit many other applications, such as information extraction, schema mapping, and information integration.

6.1.3 WOE Contributions

WOE is the first system that autonomously transfers knowledge from random editors' effort of collaboratively editing Wikipedia, to train an open information extractor. We make the following contributions:

- We present WOE, a new approach to open IE that uses Wikipedia for self-supervised learning of unlexicalized extractors. WOE achieves an F-measure between 0.572 and 0.650 on three corpora, which is between 51% and 70% higher than that of the state-of-the-art open IE system TextRunner.

- Using the same learning algorithm and features as TextRunner, we compare four different ways to generate positive and negative training examples with TextRunner's method, concluding that our Wikipedia heuristic is responsible for the bulk of WOE's improved accuracy.

- The biggest win arises from using parser features. Previous work [63] concluded that parser-based features are unnecessary for information extraction, but that work assumed the presence of lexical features. We show that abstract dependency paths are a highly informative feature when performing unlexicalized extraction.

6.2 Future Work

Although the three systems presented in this dissertation have made substantial contributions, each has remaining weaknesses that can be further improved. Also, the experience gained through working with these systems suggests several overall new directions.

6.2.1 KYLIN

For an initial prototype, KYLIN performs quite well. But there are numerous directions for improvement. Many of KYLIN's components are simple baseline implementations, because we wished an end-to-end system. We wish to apply learning to the problem of document

classification, consider more sophisticated ways of combining heuristics (e.g., stacked metalearning), test on more cases, make the result public (e.g., as a Firefox extension), and other improvements. In the longer term, we will investigate the following directions:

- **Improving Extractor Learning:** There are several ways to improve KYLIN's extractors. Hoffmann *et al.* introduced several dynamic lexicon features created from Web lists to help train extractors for Wikipedia infobox relations [61]. They show that these features dramatically improved extractors' performance. Another potentially useful feature set are deep parsing features. For simplicity, current KYLIN trains individual extractor for each relation separately. A better solution is to jointly learn multiple extractors. For example, "birthDate" and "birthPlace" attributes often appear together, and information about one attribute can help to better identity the other.

- **Exploiting Other Data Sources:** Some sources besides Wikipedia, such as Freebase, could be used to create an additional training dataset via self-supervision. For example, Mintz *et al.* consider all sentences containing both the subject and object of a Freebase record as matching sentences [78]. This will enlarge the training dataset while potentially increasing the level of noise. We have tried to run KYLIN on selected Web pages to retrieve information when it is unable to extract necessary information from a Wikipedia article, but more much needs to be done along this direction.

- **Information Verification:** Besides automatic information extraction, KYLIN should also be able to verify the correctness of the extracted information so that inconsistency can be correctly resolved. An intuitive way is utilizing outside Web knowledge such as Google indices.

- **Topic Discovery:** As the largest collaborative encyclopedia, Wikipedia should extend its coverage as much as possible. But as Wikipedia grows, it is getting harder and harder for users to identify missing or incomplete articles. KYLIN should be able

to identify these topics, and facilitate users' editing (for example, pointing users to useful information sources outside Wikipedia).

6.2.2 KOG

KOG achieves satisfying performance on Wikipedia Infobox Ontology generation. In the future we intend to use the ontology to develop an improved query interface for Wikipedia and the Web. Combining KYLIN with KOG is an obvious first step. We also anticipate an inference scheme which combines multiple facts to answer a broader range of questions. There are also several ways to improve KOG itself, including improved word sense disambiguation and extending our joint-inference approach to include schema mapping. In the longer term, the following are three important directions for future work:

- **Class and Relation Invention:** The classes and relations in KOG's ontology are confined by those defined in existing Wikipedia infoboxes. However, the Web contains a much broader set of topics that keep evolving as well. Therefore it is crucial for KOG to include a set of concept- and relation-induction mechanisms. Most work has studied concept and relation invention in isolation [18, 83]. We suspect a joint inference framework, such as SNE in [64] and OntoUSP in [91], is superior.

- **Selectional Preference:** Selectional preferences encode the set of admissible argument values for a relation. An ontology with probabilistic selectional preference has the potential to improve the performance of a wide range of NLP tasks such as information extraction, semantic role labeling and word-sense disambiguation. Ritter *et al.* applied the LinkLDA model to automatically compute selectional preferences based on a corpus of extracted triples by TextRunner [95]. A similar approach, like Labeled-LDA [92], could be applied by KOG, given the semantic tags of some arguments and relations are known according to Wikipedia infobox templates.

- **Ontology Alignment and Integration:** Multiple ontologies on the same domain are likely to present concepts in a variety of ways (e.g., "substance" in one ontology is the same as "drug" in another), and at different levels of detail (e.g., IMDB has more

details on movies than those included in Wikipedia infoboxes). KOG should be able to align and integrate these ontologies to create a consistent and richer representation. This also provides a chance for these ontologies to correct errors made by the others. One possible solution is to maintain a single giant back-bone ontology, and keeps merging newly discovered ontolgoies into it; another way out is to create probabilistic mappings between multiple ontologies. It is unclear which approach might be better.

6.2.3 WOE

In the future, we plan to run WOE over the billion document CMU ClueWeb09 corpus to compile a giant knowledge base for distribution to the NLP community. We also wish to combine WOE^{parse} with WOE^{pos} (*e.g.*, with voting) to produce a system which maximizes precision at low recall. In the longer term, there are several interesting future work for WOE:

- **Combining Lexicalized and Open Extraction:** WOE only uses unlexicalized features when learning open extractors. We are also interested in merging lexicalized and open extraction methods; the use of some domain-specific lexical features might help to improve WOE's practical performance, but the best way to do this is unclear.

- **N-ary Relation Extraction:** As most IE systems, WOE focuses on extracting facts in triple format (*i.e.* $\langle subject, predicate, object \rangle$) from natural-language texts. However, besides "subject" and "object", many facts involve more arguments such as time and location. WOE should be able to perform n-ary relation extraction to maintain the intact information.

- **Semantifying Extracted Tuples:** The extracted tuples by WOE needs further semantifying to become more useful to other applications. These include integration with a Web-scale named-entity recognizer, developing the ability to co-refer underspecified entities (e.g. $\langle he, founded, Microsoft \rangle$ refers to "Bill Gates"), disambiguating vague entity mentions (e.g., does $\langle Tom, likes, jaguar \rangle$ refer to the car or the animal?),

and normalizing the semantics of predicates (e.g., $\langle Dan, was\ born\ in, Boston\rangle$ means the "birthPlace" relation).

6.2.4 Overall New Directions

The work in this dissertation also suggests some interesting directions that are not directly tied to the three systems:

- **Multi-lingual Extraction:** We have focused on extracting information from texts in English. However, an ideal machine reading system should cover arbitrary languages. The rapid globalization of Wikipedia is generating a parallel, multi-lingual corpus of unprecedented scale, where pages for the same topic in many different languages are explicitly linked to each other. This characteristic makes Wikipedia a fantastic source for multi-lingual information extraction. Adar *et al.* proposed the Ziggurat system which automatically complete infoboxes by leveraging information among articles in different languages [6], but much more can be done.

- **Textual Inference:** We assume that all target information is stated explicitly within a single sentence, but many facts are implicitly embedded across multiple sentences. To harvest such kind of information, textural inference is necessary. A typical example is shown by Schoenmackers *et al.* in [97]: few sentences stating directly that *"Kale prevents osteoporosis."*; a system, like their HOLMES, must infer from multiple sentences, such as *"Kale contains calcium."* and *"Calcium prevents osteoporosis."*, to get this fact.

- **Human Computing:** We tried to achieve total autonomy when developing KYLIN, KOG, and WOE. Although these systems have satisfying performance, their precisions are not high enough for some practical applications. For example, although KYLIN's average precision of mid-80s percent is considered success, it is unacceptable in Wikipedia. One effective way to fill the gap is exploiting human computing — relying on human effort to verify / correct results output by machines. We tested this idea in [62] where users are presented with KYLIN's extractions for verification.

The experiments shew that the benefits are mutual: KYLIN greatly facilitates users' editing, and users' input and feedback help KYLIN to learn better extractors. There are many more opportunities along this direction. For example, how to apply active learning to maximize the utility of human effort? what's the best strategy to use human computing service like Amazon Mechanical Turk? All of these are interesting and challenging questions.

6.3 Parting Thoughts

The World Wide Web contains virtually unlimited amount of information embedded in natural-language texts. As the long-standing goal of AI and NLP, machine reading aims to automatically distill the information from texts and make them accessible to software agents. In this dissertation, we develop three systems (KYLIN, KOG, and WOE), which demonstrate the promise of bootstrapping from Wikipedia towards addressing the challenges of Web-scale machine reading and reveals exciting directions for future works.

BIBLIOGRAPHY

[1] http://wordnet.princeton.edu/.

[2] http://opennlp.sourceforge.net/.

[3] http://nlp.stanford.edu/downloads/lex-parser.shtml.

[4] http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[5] Sisay Fissaha Adafre and Maarten de Rijke. Discovering missing links in wikipedia. In *Proceedings of the Workshop on Link Discovery, KDD05*, 2005.

[6] E. Adar, M. Skinner, and D. Weld. Information arbitrage in multi-lingual Wikipedia. In *WSDM*, 2009.

[7] B. T. Adler and L. de Alfaro. A content-driven reputation system for the wikipedia. In *Proceedings of WWW07*, 2007.

[8] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *ICDL*, 2000.

[9] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[10] Alan Akbik and Jügen Broß. Wanderlust: Extracting Semantic Relations from Natural Language Text Using Dependency Grammar Patterns. In *WWW Workshop*, 2009.

[11] S. Auer, C. Bizer, J. Lehmann, G. Kobilarov, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of ISWC07*, 2007.

[12] Sören Auer and Jens Lehmann. What have Innsbruck and Leipzig in common? Extracting semantics from wiki content. In *ESWC07*, 2007.

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

[14] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the Web. In *Proceedings of IJCAI07*, 2007.

[15] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the Web. In *Proceedings of IJCAI07*, 2007.

[16] Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *ACL*, 2008.

[17] Kedar Bellare and Andrew McCallum. Learning extractors from unlabeled text using relevant databases. In *IIWeb*, 2007.

[18] Kedar Bellare, Partha Pratim Talukdar, Giridhar Kumaran, Fernando Pereira, Mark Liberman, Andrew McCallum1, and Mark Dredze. Lightly-Supervised Attribute Extraction for Web Search. In *NIPS*, 2007.

[19] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[20] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *Proceedings of ICDE05*, 2005.

[21] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, 1998.

[22] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Relational Duality: Unsupervised Extraction of Semantic Relations between Entities on the Web. In *WWW*, 2010.

[23] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[24] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proceedings of EMNLP02*, 2002.

[25] R. Bunescu and R.Mooney. A shortest path dependency kernel for relation extraction. In *HLT/EMNLP*, 2005.

[26] Razvan C. Bunescu and Raymond J. Mooney. Subsequence kernels for relation extraction. In *NIPS*, 2005.

[27] Razvan C. Bunescu and Raymond J. Mooney. Learning to extract relations from the web using minimal supervision. In *ACL*, 2007.

[28] Michael J. Cafarella, Dan Suciu, and Oren Etzioni. Navigating extracted data with schema discovery. In *Proceedings of WebDB07*, 2007.

[29] 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 *AAAI*, 2010.

[30] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*, 2005.

[31] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *Proceedings of WWW04*, 2004.

[32] P. Cimiano, G. Ladwig, and S. Staab. Gimme' the context: Context-driven automatic semantic annotation with c-pankow. In *Proceedings of WWW05*, 2005.

[33] P. Cimiano and S. Staab. Learning concept hierarchies from text with a guided agglomerative clustering algorithm. In *Proceedings of Workshop on Learning and Extending Lexical Ontologies with Machine Learning Methods, ICML05*, 2005.

[34] Philipp Cimiano, Aleksander Pivk, Lars Schmidt-Thieme, and Steffen Staab. Learning taxonomic relations from heterogeneous sources of evidence. *Ontology Learning from Text: Methods, Evaluation and Applications*, 2005.

[35] P. Clark and B. Porter. Building concept representations from reusable components. In *Proceedings of AAAI97*, 1997.

[36] Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting redundancy in question answering. In *Proceedings of SIGIR01*, 2001.

[37] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *AAAI*, 1998.

[38] W. Dakka and S. Cucerzan. Augmenting wikipedia with named entity tags. In *Proceedings of IJCNLP 2008*, 2008.

[39] Dmitry Davidov and Ari Rappoport. Unsupervised Discovery of Generic Relationships Using Pattern Clusters and its Evaluation by Automatically Generated SAT Analogy Questions. In *ACL*, 2008.

[40] Dmitry Davidov, Ari Rappoport, and Moshe Koppel. Fully Unsupervised Discovery of Concept-Specific Relationships by Web Mining. In *ACL*, 2007.

[41] Marie-Catherine de Marneffe and Christopher D. Manning. Stanford typed dependencies manual, 2008. http://nlp.stanford.edu/downloads/lex-parser.shtml.

[42] R. de Salvo Braz, R. Girju, V. Punyakanok, D. Roth, and M. Sammons. An inference model for semantic entailment in natural language. In *Proceedings of AAAI05*, 2005.

[43] P. DeRose, X. Chai, B. Gao, W. Shen, A. Doan, P. Bohannon, and J. Zhu. Building community wikipedias: A human-machine approach. In *Proceedings of ICDE08*, 2008.

[44] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. Tomlin, and J. Y. Zien. Semtag and Seeker: bootstrapping the Semantic Web via automated semantic annotation. In *Proceedings of WWW03*, 2003.

[45] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine, Special Issue on Semantic Integration*, 2005.

[46] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[47] D. Downey, O. Etzioni, and S. Soderland. A probabilistic model of redundancy in information extraction. In *Procs. of IJCAI 2005*, 2005.

[48] Doug Downey, Stefan Schoenmackers, and Oren Etzioni. Sparse information extraction: Unsupervised language models to the rescue. In *Proceedings of ACL07*, 2007.

[49] Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. Web question answering: Is more always better? In *Proceedings of SIGIR02*, 2002.

[50] Benjamin Van Durme and Lenhart K. Schubert. Open knowledge extraction using compositional language processing. In *STEP*, 2008.

[51] 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.

[52] Oren Etzioni, Michele Banko, and Stephen Soderland Daniel S. Weld. Open information extraction from the Web. In *CACM*, 2008.

[53] Evgeniy Gabrilovich and Shaul Markovitch. Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge. In *Proceedings of AAAI06*, 2006.

[54] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of IJCAI07*, 2007.

[55] Junfei Geng and Jun Yang. Autobib: Automatic extraction of bibliographic information on the Web. In *IDEAS*, 2004.

[56] Alon Y. Halevy, Oren Etzioni, AnHai Doan, Zachary G. Ives, Jayant Madhavan, Luke McDowell, and Igor Tatarinov. Crossing the structure chasm. In *Proceedings of CIDR03*, 2003.

[57] Bin He and K. Chang. Statistical schema matching across web query interfaces. In *Proceedings of SIGMOD03*, 2003.

[58] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING92*, 1992.

[59] M. Hepp, K. Siorpaes, and D. Bachlechner. Harvesting wiki consensus: Using wikipedia entries as vocabulary for knowledge management. *IEEE Internet Computing*, 11(5):54–65, 2007.

[60] Aurelie Herbelot and Ann Copestake. Acquiring ontological relationships from wikipedia using rmrs. In *Proceedings of Workshop on Web content Mining with Human Language Technologies, ISWC06*, 2006.

[61] R. Hoffmann, C. Zhang, and D. Weld. Learning 5000 relational extractors. In *ACL*, 2010.

[62] Raphael Hoffmann, Saleema Amershi, Kayur Patel, Fei Wu, James Fogarty, and Daniel S. Weld. Amplifying community content creation using mixed-initiative information extraction. In *CHI*, 2009.

[63] Jing Jiang and ChengXiang Zhai. A systematic exploration of the feature space for relation extraction. In *HLT/NAACL*, 2007.

[64] S. Kok and P. Domingos. Extracting semantic networks from text via relational clustering. In *ECML*, 2008.

[65] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, and P. Domingos. The alchemy system for statistical relational AI, technical report, university of washington, 2006.

[66] C. T. Kwok, O. Etzioni, and D. Weld. Scaling question answering to the Web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.

[67] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of WWW01*, 2001.

[68] Dekang Lin and Patrick Pantel. DIRT @SBT@discovery of inference rules from text. In *KDD*, pages 323–328, 2001.

[69] A. Gangemi M. Ciaramita. Unsupervised learning of semantic relations between concepts of a molecular biology ontology. In *IJCAI*, 2005.

[70] Bill MacCartney and Christopher D. Manning. Natural logic for textual inference. In *Workshop on Textual Entailment and Paraphrasing, ACL 2007*, 2007.

[71] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *Proceedings of ICDE05*, 2005.

[72] Cynthia Matuszek, Michael J. Witbrock, Robert C. Kahlert, John Cabral, David Schneider, Purvesh Shah, and Douglas B. Lenat. Searching for common sense: Populating cyc from the web. In *AAAI*, pages 1430–1435, 2005.

[73] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[74] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. In *http://mallet.cs.umass.edu*, 2002.

[75] Luke K. McDowell and Michael Cafarella. Ontology-driven information extraction with ontosyphon. In *Proceedings of ISWC06*, 2006.

[76] Ron Meir and Gunnar Rätsch. An introduction to boosting and leveraging. *Journal of Artificial Intelligence Research*, Advanced Lectures on Machine Learning:118–183, 2003.

[77] David Milne, Ian H. Witten, and David Nichols. A knowledge-based search engine powered by wikipedia. In *Proceedings of CIKM07*, 2007.

[78] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL-IJCNLP*, 2009.

[79] T. H. Kotaro Nakayama and S. Nishio. Wikipedia link structure and text mining for semantic relation extraction. In *CEUR Workshop*, 2008.

[80] Dat P.T Nguyen, Yutaka Matsuo, and Mitsuru Ishizuka. Exploiting syntactic and semantic information for relation extraction from wikipedia. In *IJCAI07-TextLinkWS*, 2007.

[81] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *Proceedings of Workshop on Machine Learning for Information Filtering, IJCAI99*, 1999.

[82] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, pages 169–198, 1999.

[83] M. Pasca and B. V. Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *ACL*, 2008.

[84] Marius Pasca. Turning Web Text and Search Queries into Factual Knowledge: Hierarchical Class Attribute Extraction. In *AAAI*, 2008.

[85] S. Patwardhan and E. Riloff. Effective information extraction with semantic affinity patterns and relevant regions. In *EMNLP*, 2007.

[86] Fuchun Peng and Andrew McCallum. Accurate Information Extraction from Research Papers using Conditional Random Fields. In *HLT-NAACL*, 2004.

[87] Simone Paolo Ponzetto and Michael Strube. Deriving a large scale taxonomy from wikipedia. In *Proceedings of AAAI07*, 2007.

[88] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of AAAI06*, 2006.

[89] Hoifung Poon, Janara Christensen, Pedro Domingos, Oren Etzioni, Raphael Hoffmann, Chloe Kiddon, Thomas Lin, Xiao Ling, Mausam, Alan Ritter, Stefan Schoenmackers, Stephen Soderland, Dan Weld, Fei Wu, and Congle Zhang. Machine reading at the university of washington. In *NAACL Workshop FAM-LbR*, 2010.

[90] Hoifung Poon and Pedro Domingos. Joint Inference in Information Extraction. In *AAAI*, 2008.

[91] Hoifung Poon and Pedro Domingos. Unsupervised ontological induction from text. In *ACL*, 2010.

[92] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. Labeled lda: a supervised topic model for credit attribution in multi-labeled corpora. In *EMNLP*, 2009.

[93] M. Richardson and P. Domingos. Markov logic networks. In *Machine Learning*, 2006.

[94] E. Riloff and J. Shepherd. A corpus-based approach for building semantic lexicons. In *Proceedings of EMNLP97*, Providence, RI, 1997.

[95] Alan Ritter, Mausam, and Oren Etzioni. A Latent Dirichlet Allocation method for Selectional Preferences. In *ACL*, 2010.

[96] D. Sanchez and A. Moreno. Web-scale taxonomy learning. In *Proceedings of Workshop on Extending and Learning Lexical Ontologies using Machine Learning, ICML05*, 2005.

[97] Stefan Schoenmackers, Oren Etzioni, and Daniel Weld. Scaling textual inference to the Web. In *EMNLP*, 2008.

[98] Y. Shinyama and S. Sekine. Preemptive information extraction using unristricted relation discovery. In *HLT-NAACL*, 2006.

[99] P. Singla and P. Domingos. Discriminative training of markov logic networks. In *Proceedings AAAI05*, 2005.

[100] Rion Snow, Daniel Jurafsky, and A. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of ACL06*, 2006.

[101] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005.

[102] C. Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the 3rd Berkeley Symposium on Mathematical Statistics and Probability)*, pages 197–206. University of California Press, 2002.

[103] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge - unifying WordNet and Wikipedia. In *Proceedings of WWW07*, 2007.

[104] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R Studer. Semantic wikipedia. In *Proceedings of WWW06*, 2006.

[105] P. Velardi, R. Navigli, A. Cucchiarelli, and F. Neri. Evaluation of ontolearn, a methodology for automatic learning of domain ontologies. *Ontology Learning and Population*, 2005.

[106] Mengqiu Wang. A Re-examination of Dependency Path Kernels for Relation Extraction. In *IJCNLP*, 2008.

[107] C. Welty, J. Fan, D. Gondek, and A. Schlaikjer. Large scale relation detection. In *FAM-LbR at NAACL*, 2010.

[108] F. Wu and D. Weld. Autonomouslly semantifying wikipedia. In *Proceedings of CIKM07*, Lisbon, Porgugal, 2007.

[109] F. Wu and D. Weld. Autonomouslly semantifying wikipedia. In *Proceedings of CIKM07*, Lisbon, Porgugal, 2007.

[110] W. Wu, A. Doan, C. Yu, and W. Meng. Bootstrapping domain ontology for Semantic Web services from source web sites. In *Proceedings of Workshop on Technologies for E-Services, VLDB05*, 2005.

[111] K. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proceedings of SIGCHI03*, 2003.

[112] Min Zhang, Jie Zhang, Jian Su, and Guodong Zhou. A composite kernel to extract relations between entities with both flat and structured features. In *ACL*, 2006.

[113] Shubin Zhao and Ralph Grishman. Extracting relations with integrated information using kernel methods. In *ACL*, 2005.

[114] Jun Zhu, Zaiqing Nie, Xiaojiang Liu, Bo Zhang, and Ji-Rong Wen. Statsnowball: a statistical approach to extracting entity relationships. In *WWW*, 2009.

Appendix A

DATA FOR DISTRIBUTION

The following datasets are available at the following site:

http://ai.cs.washington.edu/projects/intelligent-wikipedia.

- The "pseudo-manually" labeled dataset for testing KOG's subsumption detection performance. It contains:
  - Mappings from 406 infobox classes to WordNet nodes based on DBpedia
  - Positive and negative examples of subsumption relations
- The "schema-mapping" test set for KOG, which contains 10 pairs of parent/child classes with labeled attribute mappings in between.
- The Wikipedia Infobox Ontology created by KOG based on the July 2007 English version of Wikipedia. It contains:
  - Infobox Schemata: cleaned infobox classes and attributes
  - ISA Tree: ISA tree spanning the infobox classes
  - Schema Mapping: attribute mappings between parent/child classes
- The three test datasets for WOE, which are created by randomly selecting 300 sentences from each of the following corpora: WSJ from Penn Treebank, Wikipedia, and the general Web. For each sentence, ground-truth triples (with variations for $\texttt{arg}_1$, $\texttt{arg}_2$ and `rel`) are labeled.
- The 50SENT set, which is created by sampling 50 Wikipedia sentences for five relations (*person.birthPlace, person.spouse, university.establishment, university.nickname,* and

county.water_square_mile) — 10 sentences per relation. Each sentences is annotated with a name/value pair for certain target relation.

- The extraction pattern database DB_p constructed by WOE. Each pattern is associated with a frequency value.