Open Information Extraction using Wikipedia

(An updated and corrected version of our ACL-2010 paper)

Fei Wu
University of Washington
Seattle, WA, USA
wufei@cs.washington.edu

Daniel S. Weld
University of Washington
Seattle, WA, USA
weld@cs.washington.edu

Abstract

Information-extraction (IE) systems seek to distill semantic relations from natural-language text, but most systems use supervised learning of relation-specific examples and are thus limited by the availability of training data. *Open* IE systems such as TextRunner, on the other hand, aim to handle the unbounded number of relations found on the Web. But how well can these open systems perform?

This paper presents WOE, an open IE system which improves dramatically on TextRunner's precision and recall. The key to WOE's performance is a novel form of self-supervised learning for open extractors — using heuristic matches between Wikipedia infobox attribute values and corresponding sentences to construct training data. Like TextRunner, WOE's extractor eschews lexicalized features and handles an unbounded set of semantic relations. WOE can operate in two modes: when restricted to POS tag features, it runs as quickly as TextRunner, but when set to use dependency-parse features its precision and recall rise even higher.

1 Introduction

The problem of information-extraction (IE), generating relational data from natural-language text, has received increasing attention in recent years. A large, high-quality repository of extracted tuples can potentially benefit a wide range of NLP tasks such as question answering, ontology learning, and summarization. The vast majority of IE work uses supervised learning of relation-specific examples. For example, the WebKB project (Craven et al., 1998) 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 and are unlikely to scale to the thousands of relations found in text on the Web.

An alternative paradigm, *Open* IE, pioneered by the TextRunner system (Banko et al., 2007) and the "preemptive IE" in (Shinyama and Sekine, 2006), 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 paper 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*[1] attribute values to corresponding sentences (as done in Kylin (Wu and Weld, 2007) and Luchs (Hoffmann et al., 2010)), 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 runs as quickly as Textrunner, but when set to use dependency-parse features its precision and recall rise even higher. We present a thorough experimental evaluation, making the following contributions:

- We present WOE, a new approach to open IE that uses Wikipedia for self-supervised learn-

[1] An infobox is a set of tuples summarizing the key attributes of the subject in a Wikipedia article. For example, the infobox in the article on "Sweden" contains attributes like *Capital, Population* and *GDP*.

ing of unlexicalized extractors. Compared with TextRunner (the state of the art) on three corpora, WOE yields between 51% and 70% improved F-measure — generalizing well beyond Wikipedia.

- Using the same learning algorithm and features as TextRunner, we compare four different ways to generate positive and negative training data 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 (Jiang and Zhai, 2007) 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.

2 Problem Definition

An open information extractor is a function from a document, d, to a set of triples, $\{\langle \texttt{arg}_1, \texttt{rel}, \texttt{arg}_2 \rangle\}$, where the args 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. In this paper, 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 wish to learn an open extractor *without direct supervision*, *i.e.* without annotated training examples or hand-crafted patterns. Our input is Wikipedia, a collaboratively-constructed encyclopedia[2]. As output, WOE produces an unlexicalized and relation-independent open extractor. Our objective is an extractor which generalizes beyond Wikipedia, handling other corpora such as the general Web.

3 Wikipedia-based Open IE

The key idea underlying WOE is the automatic construction of training examples by heuristically matching Wikipedia infobox values and corresponding text; these examples are used to generate an unlexicalized, relation-independent (open) extractor. As shown in Figure 1, WOE has three main components: preprocessor, matcher, and learner.

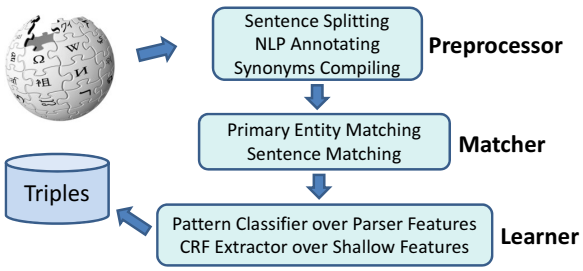


Figure 1: Architecture of WOE.

3.1 Preprocessor

The preprocessor converts the raw Wikipedia text into a sequence of sentences, attaches NLP annotations, and builds synonym sets for key entities. The resulting data is fed to the matcher, described in Section 3.2, which generates the training set.

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

NLP Annotation: As we discuss fully in Section 4 (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 (Wu and Weld, 2007; Nakayama and Nishio, 2008), the preprocessor uses Wikipedia redirection pages and back-

[2]We also use DBpedia (Auer and Lehmann, 2007) as a collection of conveniently parsed Wikipedia *infoboxes*

ward 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 "Massachusetts 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.

3.2 Matcher

The matcher constructs training data for the learner component by heuristically matching attribute-value pairs from Wikipedia articles containing infoboxes with corresponding sentences in the article. Given the article on "Stanford University," for example, the matcher should associate $\langle \texttt{established}, 1891 \rangle$ with the sentence "The university was founded in 1891 by ..." Given a Wikipedia page with an infobox, the matcher iterates through all its attributes looking for a unique sentence that contains references to both the subject of the article and the attribute value; these noun phrases will be annotated $\texttt{arg}_1$ and $\texttt{arg}_2$ in the training set. The matcher considers a sentence to contain the attribute value if the value or its synonym is present. Matching the article subject, however, is more involved.

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 (Wu and Weld, 2007; Nguyen et al., 2007):

- Full match: strings matching the full name of the entity are selected.
- Synonym set match: strings appearing in the entity's synonym set are selected.
- 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.
- 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.
- 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 infobox attribute value in the parser dependency graph.

Matching Sentences: The matcher seeks a *unique* sentence to match the attribute value. To produce the best training set, the matcher performs three filterings. 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 (Auer and Lehmann, 2007) 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.

3.3 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 field (CRF) to output certain text between noun phrases when the text denotes such a relation. Neither extractor uses individual words or lexical information for features.

3.3.1 Extraction with Parser Features

Despite some evidence that parser-based features have limited utility in IE (Jiang and Zhai, 2007), 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 are collapsed to get direct dependencies between content words. As

noted in (de Marneffe and Manning, 2008), 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:

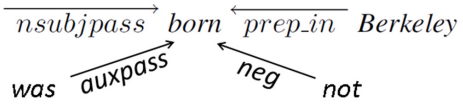
nsubjpass(born-4, Dan-1)
auxpass(born-4, was-2)
neg(born-4, not-3)
prep_in(born-4, Berkeley-6)

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 $\overrightarrow{nsubjpass}$ *born* $\overleftarrow{prep\_in}$ *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* $\overrightarrow{nsubjpass}$ *born* $\overleftarrow{prep\_in}$ *Berkeley*" for example, its generalized-corePath is "*N* $\overrightarrow{nsubjpass}$ *V* $\overleftarrow{prep}$ *N*". We call such a generalized-corePath an extraction pattern. In total, WOE builds a database (named DB_p) of 15,333 distinct patterns and each pattern p is associated with a frequency — the number of matching sentences containing p. Specifically, 185 patterns have $f_p \geq 100$ and 1929 patterns have $f_p \geq 5$.

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]:

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

where f_{max} (50,259 in this paper) 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.

Take the previous sentence "*Dan was not born in Berkeley*" for example. 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 $\langle Dan, wasNotBornIn, Berkeley \rangle$. As shown in the experiments on three corpora, WOE^{parse} achieves an F-measure which is between 51% to 70% greater than TextRunner's.

3.3.2 Extraction with Shallow Features

WOE^{parse} has a dramatic performance improvement over TextRunner. However, the improvement comes at the cost of speed — TextRunner

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

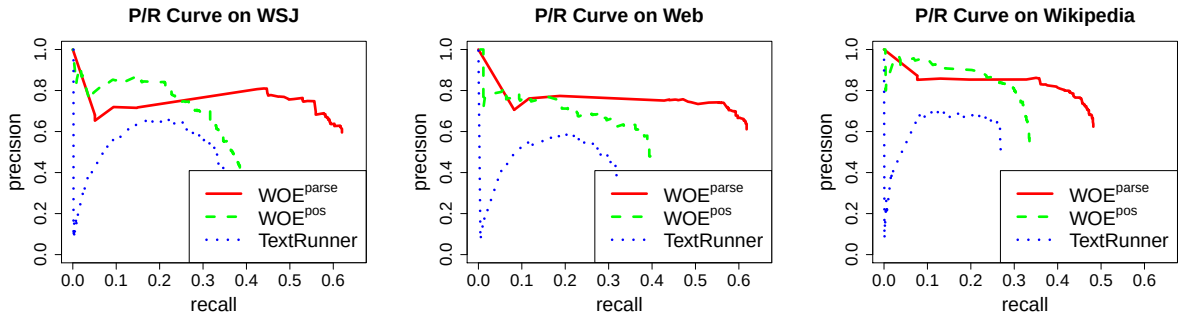


Figure 2: WOE^{pos}performs better than TextRunner, especially on precision. WOE^{parse}dramatically improves performance, especially on recall.

runs about $30X$ faster by only using shallow features. 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 their generalized-CorePaths are not in DB_p.

WOE^{pos} uses the same learning algorithm and selection of features as TextRunner: a two-order CRF chain model is trained with the Mallet package (McCallum, 2002). 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.

4 Experiments

We used three corpora for experiments: WSJ from Penn Treebank, Wikipedia, and the general Web. For each dataset, we randomly selected 300 sentences. 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.

4.1 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 are going to answer the following questions: 1) How do these systems perform against each other? 2) How does performance vary w.r.t. sentence length? 3) How does extraction speed vary w.r.t. sentence length?

Overall Performance Comparison

The detailed P/R curves are shown in Figure 2. 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 3. We can see that:

- WOE^{pos} is better than TextRunner, especially on precision. This is due to better training data from Wikipedia via self-supervision. Section 4.2 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 sentences. In particular, WOE^{parse} outputs 2.1 triples per sentence on average, while WOE^{pos} outputs 1.7 and TextRunner outputs 1.5.

Note that we measure TextRunner's precision & recall differently than (Banko et al., 2007) did. 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, $\texttt{arg}_2$ is a proper noun or date, and

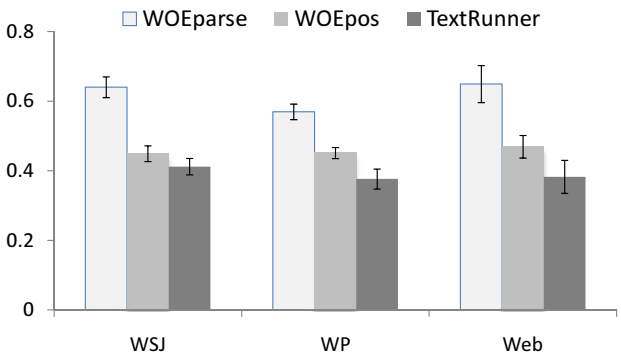


Figure 3: WOE^{pos} achieves an F-measure, which is between 9% and 23% better than TextRunner's. WOE^{parse} achieves an improvement between 51% and 70% over TextRunner. The error bar indicates one standard deviation.

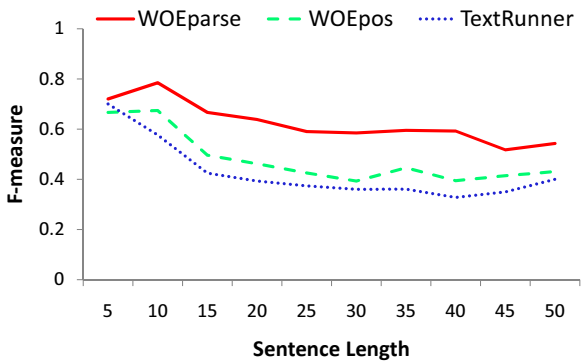


Figure 4: WOE^{parse}'s F-measure decreases more slowly with sentence length than WOE^{pos} and TextRunner, due to its better handling of difficult sentences using parser features.

the frequency of `rel` is over a threshold. Our experiments show that focussing on concrete triples generally improves precision at the expense of recall.[4] Of course, one can apply a concreteness filter to any open extractor in order to trade recall for precision.

The extraction errors by WOE^{parse} can be categorized into four classes. We illustrate them with the WSJ corpus. In total, WOE^{parse} got 85 wrong extractions on WSJ, and they are 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%).

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 $\langle Sources, sold, theCompany\rangle$ extracted from the sentence "Sources said he sold the company", where "Sources" is wrongly treated as the subject of the object clause. A sample error of the second type is $\langle thisYear, willStarIn, theMovie\rangle$ extracted from the sentence "Coming up this year, 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=0.059, these two types of errors account for 70% of all errors.

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

We tested how extractors' performance varies with sentence length; the results are shown in Figure 4. 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, showing that TextRunner and WOE^{pos}'s extraction time grows approximately linearly with sentence

[4] For example, consider the Wikipedia corpus. From our 300 test sentences, TextRunner extracted 413 triples (at 62.4% precision) but only extracted 22 concrete triples (with 81.8% precision).

[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.

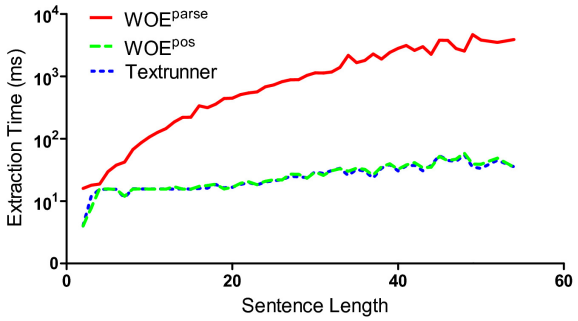


Figure 5: Textrnner and WOE^{pos}'s running time seems to grow linearly with sentence length, while WOE^{parse}'s time grows quadratically.

length, while WOE^{parse}'s extraction time grows quadratically ($R^2 = 0.935$) due to its reliance on parsing.

4.2 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 221,205 positive examples based on the matching sentence set[6]. All extractors are trained using about the same number of positive and negative examples. In contrast, TextRunner was trained 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 6, 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.

[6]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.

4.3 Design Desiderata of WOE^{parse}

There are two interesting design choices in WOE^{parse}: 1) whether to require $\texttt{arg}_1$ to appear before $\texttt{arg}_2$ (denoted as 1<2) in the sentence; 2) whether to allow corePaths to contain prepositional phrase (PP) attachments (denoted as *PPa*). We tested how they affect the extraction performance; the results are shown in Figure 7.

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<2) yields a smaller improvement in precision with small loss in recall. Take the WSJ corpus for example: setting 1<2 and $\overline{PPa}$ achieves a precision of 0.748 (with recall of 0.556). By changing 1<2 to 1~2, the precision decreases to 0.730 (with recall of 0.591). By changing $\overline{PPa}$ to *PPa* and keeping 1<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<2 and $\overline{PPa}$ as default in WOE^{parse} as a logical consequence of our preference for high precision over high recall.

4.3.1 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 (Charniak and Johnson, 2005), and the gold parses from the Penn Treebank. The Stanford Parser is used to derive dependencies from CJ50 and gold parse trees. Figure 8 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.

5 Related Work

Open or Traditional Information Extraction: Most existing work on IE is relation-specific. Occurrence-statistical models (Agichtein and Gravano, 2000; M. Ciaramita, 2005), graphical models (Peng and McCallum, 2004; Poon and Domingos, 2008), and kernel-based methods (Bunescu and R.Mooney, 2005) have been studied. Snow et al. (Snow et al., 2005) utilize WordNet to learn dependency path patterns for extracting the hypernym relation from text. Some seed-based frameworks are proposed for open-domain extraction (Pasca, 2008; Davidov et al., 2007; Davidov and Rappoport, 2008). These works focus

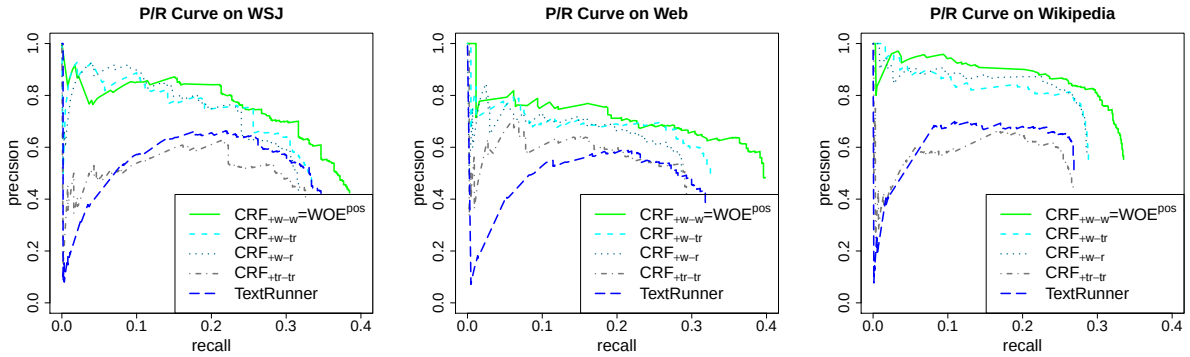


Figure 6: Matching sentences with Wikipedia infoboxes results in better training data than the hand-written rules used by TextRunner.

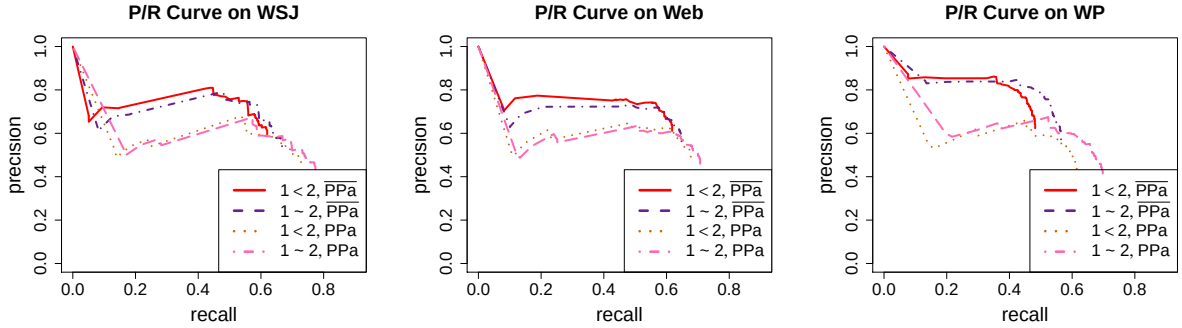


Figure 7: Filtering prepositional phrase attachments ($\overline{PPa}$) shows a strong boost to precision, and we see a smaller boost from enforcing a lexical ordering of relation arguments ($1<2$).

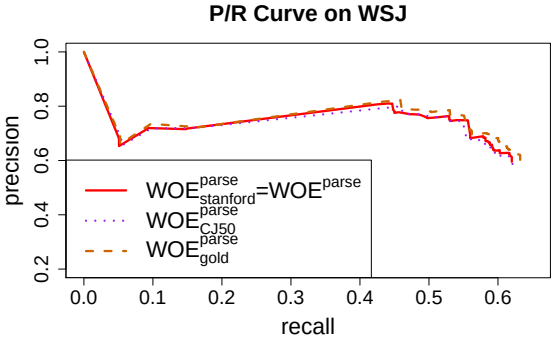


Figure 8: Although today's statistical parsers make errors, they have negligible effect on the accuracy of WOE compared to operation on gold standard, human-annotated data.

on identifying general relations such as class attributes, while open IE aims to extract relation instances from given sentences. Another seed-based system StatSnowball (Zhu et al., 2009) 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 (Shinyama and Sekine, 2006). 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. Mintz et al. (Mintz et al., 2009) 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. Matching Freebase with arbitrary sentences instead of matching Wikipedia infobox with corresponding Wikipedia articles will potentially increase the size of matched sentences at a cost of accuracy. Also, their learned extractors are relation-specific. Alan Akbik et al. (Akbik and Broß, 2009) annotated 10,000 sentences parsed with LinkGrammar and selected 46 general linkpaths as patterns for relation extraction. In contrast, WOE learns 15,333 general pat-

terns based on an automatically annotated set of 301,962 Wikipedia sentences. The KNext system (Durme and Schubert, 2008) performs open knowledge extraction via significant heuristics. Its output is knowledge represented as logical statements instead of information represented as segmented text fragments.

Information Extraction with Wikipedia: The YAGO system (Suchanek et al., 2007) extends WordNet using facts extracted from Wikipedia categories. It only targets a limited number of predefined relations. Nakayama et al. (Nakayama and Nishio, 2008) parse selected Wikipedia sentences and perform extraction over the phrase structure trees based on several handcrafted patterns. Wu and Weld proposed the KYLIN system (Wu and Weld, 2007; Wu et al., 2008) which has the same spirit of matching Wikipedia sentences with infoboxes to learn extractors. However, it only works for relations defined in Wikipedia infoboxes.

Shallow or Deep Parsing: Shallow features, like POS tags, enable fast extraction over large-scale corpora (Davidov et al., 2007; Banko et al., 2007). Deep features are derived from parse trees with the hope of training better extractors (Zhang et al., 2006; Zhao and Grishman, 2005; Bunescu and Mooney, 2005; Wang, 2008). Jiang and Zhai (Jiang and Zhai, 2007) 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.

6 Conclusion

This paper 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. Comparing with TextRunner, WOE^{pos} runs at the same speed, but achieves an F-measure which is between 9% and 23% greater on three corpora; WOE^{parse} achieves an F-measure which is between 51% and 70% higher than that of TextRunner, but runs about $30X$ times slower due to its reliance on 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 (Jiang and Zhai, 2007).

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. There are several ways to further improve WOE's performance. Other data sources, 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 (Mintz et al., 2009); while they use this data to learn relation-specific extractors, one could also learn an open extractor. 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. Finally, we wish to combine WOE^{parse} with WOE^{pos} (*e.g.*, with voting) to produce a system which maximizes precision at low recall.

Acknowledgements

We thank Oren Etzioni and Michele Banko from Turing Center at the University of Washington for providing the code of their software and useful discussions. We thank Alan Ritter, Mausam, Peng Dai, Raphael Hoffmann, Xiao Ling, Stefan Schoenmackers, Andrey Kolobov and Daniel Suskin for valuable comments. We also thank anonymous reviewers for helpful suggestions. This material is based upon work supported by the WRF / TJ Cable Professorship, a gift from Google and by the Air Force Research Labora-

tory (AFRL) under prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the Air Force Research Laboratory (AFRL).

References

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

Alan Akbik and Jügen Broß. 2009. Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns. In *WWW Workshop*.

Sören Auer and Jens Lehmann. 2007. What have innsbruck and leipzig in common? extracting semantics from wiki content. In *ESWC*.

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

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

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

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

M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. 1998. Learning to extract symbolic knowledge from the world wide web. In *AAAI*.

Dmitry Davidov and Ari Rappoport. 2008. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated sat analogy questions. In *ACL*.

Dmitry Davidov, Ari Rappoport, and Moshe Koppel. 2007. Fully unsupervised discovery of concept-specific relationships by web mining. In *ACL*.

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

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

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

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

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

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

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

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

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

Marius Pasca. 2008. Turning web text and search queries into factual knowledge: Hierarchical class attribute extraction. In *AAAI*.

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

Hoifung Poon and Pedro Domingos. 2008. Joint Inference in Information Extraction. In *AAAI*.

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

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

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

Mengqiu Wang. 2008. A re-examination of dependency path kernels for relation extraction. In *IJCNLP*.

Fei Wu and Daniel Weld. 2007. Autonomouslly Semantifying Wikipedia. In *CIKM*.

Fei Wu, Raphael Hoffmann, and Danel S. Weld. 2008. Information extraction from Wikipedia: Moving down the long tail. In *KDD*.

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

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

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