URES : an Unsupervised Web Relation Extraction System

Benjamin Rosenfeld
Computer Science Department
Bar-Ilan University
Ramat-Gan, ISRAEL
grurgrur@gmail.com

Ronen Feldman
Computer Science Department
Bar-Ilan University
Ramat-Gan, ISRAEL
feldman@cs.biu.ac.il

Abstract

Most information extraction systems either use hand written extraction patterns or use a machine learning algorithm that is trained on a manually annotated corpus. Both of these approaches require massive human effort and hence prevent information extraction from becoming more widely applicable. In this paper we present URES (Unsupervised Relation Extraction System), which extracts relations from the Web in a totally unsupervised way. It takes as input the descriptions of the target relations, which include the names of the predicates, the types of their attributes, and several seed instances of the relations. Then the system downloads from the Web a large collection of pages that are likely to contain instances of the target relations. From those pages, utilizing the known seed instances, the system learns the relation patterns, which are then used for extraction. We present several experiments in which we learn patterns and extract instances of a set of several common IE relations, comparing several pattern learning and filtering setups. We demonstrate that using simple noun phrase tagger is sufficient as a base for accurate patterns. However, having a named entity recognizer, which is able to recognize the types of the relation attributes significantly, enhances the extraction performance. We also compare our approach with KnowItAll's fixed generic patterns.

1 Introduction

The most common preprocessing technique for text mining is information extraction (IE). It is defined as the task of extracting knowledge out of textual documents. In general, IE is divided into two main types of extraction tasks – *Entity tagging* and *Relation extraction*.

The main approaches used by most information extraction systems are the knowledge engineering approach and the machine learning approach. The knowledge engineering (mostly rule based) systems traditionally were the top performers in most IE benchmarks, such as MUC (Chinchor, Hirschman et al. 1994), ACE and the KDD CUP (Yeh and Hirschman 2002). Recently though, the machine learning systems became state-of-the-art, especially for simpler tagging problems, such as named entity recognition (Bikel, Miller et al. 1997), or field extraction (McCallum, Freitag et al. 2000). The general idea is that a domain expert labels the target concepts in a set of documents. The system then learns a model of the extraction task, which can be applied to new documents automatically.

Both of these approaches require massive human effort and hence prevent information extraction from becoming more widely applicable. In order to minimize the huge manual effort involved with building information extraction systems, we have designed and developed URES (Unsupervised Relation Extraction System) which learns a set of patterns to extract relations from the web in a totally unsupervised way. The system takes as input the names of the target relations, the types of its arguments, and a small set of seed instances of the relations. It then uses a large set of unlabeled documents downloaded from the Web in order to build extraction patterns. URES patterns currently have two modes of operation. One is based upon a generic shallow parser, able to extract noun phrases and their heads. Another mode builds patterns for use by TEG (Rosenfeld, Feldman et al. 2004). TEG is a hybrid rule-based and statistical IE system. It utilizes a trained labeled corpus in order to complement and enhance the performance of a relatively small set of manually-built extraction rules. When it is used with URES, the relation extraction rules and training data are not built manually but are created automatically from the URES-learned patterns. However, URES does not built rules and training data for entity extraction. For those, we use the grammar and training data we developed separately.

It is important to note that URES is not a classic IE system. Its purpose is to extract as many as possible different instances of the given relations while maintaining a high precision. Since the goal is to extract *instances* and not *mentions*, we are quite willing to miss a particular sentence containing an instance of a target relation – if the instance can be found elsewhere. In contrast, the classical IE systems extract *mentions* of entities and relations from the input documents. This difference in goals leads to different ways of measuring the performance of the systems.

The rest of the paper is organized as follows: in Section 2 we present the related work. In Section 3 we outline the general design principles of URES and the architecture of the system and then describe the different components of URES in details while giving examples to the input and output of each component. In Section 4 we present our experimental evaluation and then wrap up with conclusions and suggestions for future work.

2 Related Work

Information Extraction (IE) is a sub-field of NLP, aims at aiding people to sift through large volume of documents by automatically identifying and tagging key entities, facts and events mentioned in the text.

Over the years, much effort has been invested in developing accurate and efficient IE systems. Some of the systems are rule-based (Fisher, Soderland et al. 1995; Soderland 1999), some are statistical (Bikel, Miller et al. 1997; Collins and Miller 1998; Manning and Schutze 1999; Miller, Schwartz et al. 1999) and some are based on inductive-logic-based (Zelle and Mooney. 1996; Califf and Mooney 1998). Recent IE research with bootstrap learning (Brin 1998; Riloff and Jones 1999; Phillips and Riloff 2002; Thelen and Riloff 2002) or learning from documents tagged as relevant (Riloff 1996; Sudo, Sekine et al. 2001) has decreased, but not eliminated hand-tagged training.

Snowball (Agichtein and Gravano 2000) is an unsupervised system for learning relations from document collections. The system takes as input a set of seed examples for each relation, and uses a clustering technique to learn patterns from the seed examples. It does rely on a full fledges Named Entity Recognition system. Snowball achieved fairly low precision figures (30-50%) on relations such as merger and acquisition on the same dataset used in our experiments.

KnowItAll system is a direct predecessor of URES. It is developed at University of Washington by Oren Etzioni and colleagues (Etzioni, Cafarella et al. 2005). KnowItAll is an autonomous, domain-independent system that extracts facts from the Web. The primary focus of the system is on extracting entities (unary predicates). The input to KnowItAll is a set of entity classes to be extracted, such as “city”, “scientist”, “movie”, etc., and the output is a list of entities extracted from the Web. KnowItAll uses a set of manually-built generic rules, which are instantiated with the target predicate names, producing queries, patterns and discriminator phrases. The queries are passed to a search engine, the suggested pages are downloaded and processed with patterns. Every time a pattern is matched, the extraction is generated and evaluated using Web statistics – the number of search engine hits of the extraction alone and the extraction together with discriminator phrases. KnowItAll has also a pattern learning module (PL) that is able to learn patterns for extracting entities. However, it is unsuitable for learning patterns for relations. Hence, for extracting relations KnowItAll currently uses only the generic hand written patterns.

3 Description of URES

The goal of URES is extracting instances of relations from the Web without human supervision. Accordingly, the input of the system is limited to (reasonably short) definition of the target relations. The output of the system is a large list of relation instances, ordered by confidence. The system consists of several largely independent components. The *Sentence Gatherer* generates (e.g., downloads from the Web) a large set of sentences that may contain target instances. The *Pattern Learner* uses a small number of known seed instances to learn likely patterns of relation occurrences. The *Sentence Classifier* filters the set of sentences, removing those that are unlikely to contain instances of the target relations. The *Instance Extractor* extracts the attributes of the instances from the sentences, and generates the output of the system.

3.1 Sentence Gatherer

The Sentence Gatherer is currently implemented in a very simple way. It gets a set of keywords as input, and proceeds to download all documents that contain one of those keywords. From the documents, it extracts all sentences that contain at least one of the keywords.

The keywords for a relation are the words that are indicative of instances of the relation. The keywords are given to the system as part of the relation definition. Their number is usually small. For instance, the set of keywords for *Acquisition* in our experiments contains two words – "acquired" and "acquisition". Additional keywords (such as "acquire", "purchased", and "hostile takeover") can be added automatically by using WordNet (Miller 1995).

3.2 Pattern Learner

The task of the Pattern Learner is to learn the patterns of occurrence of relation instances. This is an inherently supervised task, because at least some occurrences must be known in order to be able to find patterns among them. Consequently, the input to the Pattern Learner includes a small set (10-15 instances) of known instances for each target relation. Our system assumes that the seeds are a part of the target relation definition. However, the seeds need not be created manually. Instead, they can be taken from the top-scoring results of a high-precision low-recall unsupervised extraction system, such as KnowItAll. The seeds for our experiments were produced in exactly this way.

The Pattern Learner proceeds as follows: first, the gathered sentences that contain the seed instances are used to generate the *positive* and *negative sets*. From those sets the pattern are learned. Then, the patterns are post-processed and filtered. We shall now describe those steps in detail.

Preparing the positive and negative sets

The positive set of a predicate (the terms *predicate* and *relation* are interchangeable in our work) consists of sentences that contain a known instance of the predicate, with the instance attributes changed to "*<AttrN>*", where N is the attribute index. For example, assuming there is a seed instance *Acquisition*(*Oracle*, *PeopleSoft*), the sentence

> *The Antitrust Division of the U.S. Department of Justice evaluated the likely competitive effects of Oracle's proposed acquisition of PeopleSoft.*

will be changed to

> *The Antitrust Division… …of <Attr1>'s proposed acquisition of <Attr2>.*

The positive set of a predicate P is generated straightforwardly, using substring search.

The negative set of a predicate consists of similarly modified sentences with known false instances of the predicate. We build the negative set as a union of two subsets. The first subset is generated from the sentences in the positive set by changing the assignment of one or both attributes to some other suitable entity. In the first mode of operation, when only a shallow parser is available, any suitable noun phrase can be assigned to an attribute. Continuing the example above, the following sentences will be included in the negative set:

> *<Attr1> of <Attr2> evaluated the likely…*
> *<Attr2> of the U.S. … …acquisition of <Attr1>.*
> etc.

In the second mode of operation, when the NER is available, only entities of the correct type get assigned to an attribute.

The other subset of the negative set contains all sentences produced in a similar way from the positive sentences of all other target predicates. We assume without loss of generality that the predicates that are being extracted simultaneously are all disjoint. In addition, the definition of each predicate indicates whether the predicate is symmetric (like "merger") or antisymmetric (like "acquisition"). In the former case, the sentences produced by exchanging the attributes in positive sentences are placed into the positive set, and in the later case – into the negative set of the predicate.

The following pseudo code shows the process of generating the positive and negative sets in detail:

```
Let $S$ be the set of gathered sentences.
For each predicate $P$
    For each $s \in S$ containing a word from Keywords($P$)
      For each known seed $P(A_1, A_2)$ of the predicate $P$
        If $A_1$ and $A_2$ are each found exactly once inside $s$
          For all entities $e_1, e_2 \in s$, such that $e2 \neq e1$, and
                  Type($e1$) = type of Attr1 of $P$, and
                  Type($e2$) = type of Attr2 of $P$
            Let $s'$ := $s$ with $e_N$ changed to "<AttrN>".
            If $e_1 = A_1$ and $e_2 = A_2$
              Add $s'$ to the PositiveSet($P$).
            Elseif $e_1 = A_2$ and $e_2 = A_1$ and symmetric($P$)
              Add $s'$ to the PositiveSet($P$).
            Else
              Add $s'$ to the NegativeSet($P$).
For each predicate $P$
   For each predicate $P_2 \neq P$
      For each sentence $s \in$ PositiveSet($P_2$)
         Put $s$ into the NegativeSet($P$).
```

Generating the patterns

The patterns for predicate *P* are generalizations of pairs of sentences from the positive set of *P*. The function *Generalize*(S_1, S_2) is applied to each pair of sentences S_1 and S_2 from the positive set of the predicate. The function generates a pattern that is the best (according to the objective function defined below) generalization of its two arguments. The following pseudo code shows the process of generating the patterns:

```
For each predicate $P$
   For each pair $S_1$, $S_2$ from PositiveSet($P$)
      Let Pattern := Generalize($S_1$, $S_2$).
      Add Pattern to PatternsSet($P$).
```

The patterns are sequences of *tokens*, *skips* (denoted *), *limited skips* (denoted *?) and *slots*. The tokens can match only themselves, the skips match zero or more arbitrary tokens, and slots match instance attributes. The limited skips match zero or more arbitrary tokens, which must not belong to entities of the types equal to the types of the predicate attributes. The *Generalize*(s_1, s_2) function takes two patterns (note, that sentences in the positive and negative sets are patterns without skips) and generates the least (most specific) common generalization of both. The function does a dynamical programming search for the best match between the two patterns (Optimal String Alignment algorithm), with the cost of the match defined as the sum of costs of matches for all elements. We use the following numbers: two identical elements match at cost 0, a token matches a skip or an empty space at cost 10, a skip matches an empty space at cost 2, and different kinds of skip match at cost 3. All other combinations have infinite cost. After the best match is found, it is converted into a pattern by copying matched identical elements and adding skips where non-identical elements are matched. For example, assume the sentences are

Toward this end, <Attr1> in July acquired <Attr2>
Earlier this year, <Attr1> acquired <Attr2> from X

After the dynamical programming-based search, the following match will be found:

Table 1 - Best Match between Sentences

Toward		(cost 10)
	Earlier	(cost 10)
this	*this*	(cost 0)
end		(cost 10)
	year	(cost 10)
,	,	(cost 0)
<Attr1>	*<Attr1>*	(cost 0)
in July		(cost 20)
acquired	*acquired*	(cost 0)
<Attr2>	*<Attr2>*	(cost 0)
	from	(cost 10)
	X	(cost 10)

at total cost = 80. The match will be converted to the pattern (assuming the NER mode, so the only entity belonging to the same type as one of the attributes is "*X*"):

*? *? *this* *? *? , *<Attr1>* *? *acquired* *<Attr2>* *? *

which becomes, after combining adjacent skips,

*? *this* *? , *<Attr1>* *? *acquired* *<Attr2>* *

Note, that the generalization algorithm allows patterns with any kind of elements beside skips, such as *CapitalWord*, *Number*, *CapitalizedSequence*, etc. As long as the costs and results of matches are properly defined, the *Generalize* function is able to find the best generalization of any two patterns. However, in the present work we stick with the simplest pattern definition as described above.

Post-processing, filtering, and scoring

The number of patterns generated at the previous step is very large. Post-processing and filtering tries to reduce this number, keeping the most useful patterns and removing the too specific and irrelevant ones.

First, we remove from patterns all "stop words" surrounded by skips from both sides, such as the word "*this*" in the last pattern in the previous subsection. Such words do not add to the discriminative power of patterns, and only needlessly reduce the pattern recall. The list of stop words includes all functional and very common English words, as well as puncuation marks. Note, that the stop words are removed only if they are surrounded by skips, because when they are adjacent to slots or non-stop words they often convey valuable information. After this step, the pattern above becomes

**? , <Attr1> *? acquired <Attr2> **

In the next step of filtering, we remove all patterns that do not contain *relevant* words. For each predicate, the list of relevant words is automatically generated from WordNet by following all links to depth at most 2 starting from the predicate keywords. For example, the pattern

** <Attr1> * by <Attr2> **

will be removed, while the pattern

** <Attr1> * purchased <Attr2> **

will be kept, because the word "*purchased*" can be reached from "*acquisition*" via synonym and derivation links.

The final (optional) filtering step removes all patterns, that contain slots surrounded by skips on both sides, keeping only the patterns, whose slots are adjacent to tokens or to sentence boundaries. Since both the shallow parser and the NER system that we use are far from perfect, they often place the entity boundaries incorrectly. Using only patterns with *anchored* slots significantly improves the precision of the whole system. In our experiments we compare the performance of anchored and unanchored patterns.

The filtered patterns are then scored by their performance on the positive and negative sets. Currently we use a simple scoring method – the score of a pattern is the number of positive matches divided by the number of negative matches plus one:

$$Score(Pattern) = \frac{|\{S \in PositiveSet : Pattern \text{ matches } S\}|}{|\{S \in NegativeSet : Pattern \text{ matches } S\}| + 1}$$

This formula is purely empirical and produces reasonable results. The threshold is applied to the set of patterns, and all patterns scoring less than the threshold (currently, it is set to 6) are discarded.

3.3 Sentence Classifier

The task of the Sentence Classifier is to filter out from the large pool of sentences produced by the Sentence Gatherer the sentences that do not contain the target predicate instances. In the current version of our system, this is only done in order to reduce the number of sentences that need to be processed by the Slot Extractor. Therefore, in this stage we just remove the sentences that do not match any of the regular expressions generated from the patterns. Regular expressions are generated from patterns by replacing slots with skips.

3.4 Instance Extractor

The task of the Instance Extractor is to use the patterns generated by the Pattern Learner on the sentences that were passed through by the Sentence Classifier. However, the patterns cannot be directly matched to the sentences, because the patterns only define the placeholders for instance attributes and cannot by themselves extract the values of the attributes.

We currently have two different ways to solve this problem – using a general-purpose shallow parser, which is able to recognize noun phrases and their heads, and using an information extraction system called TEG (Rosenfeld, Feldman et al. 2004), together with a trained grammar able to recognize the entities of the types of the predicates' attributes. We shall briefly describe the two modes of operation.

Shallow Parser mode

In the first mode of operation, the predicates may define attributes of two different types: *ProperName* and *CommonNP*. We assume that the values of the *ProperName* type are always heads of proper noun phrases. And the values of the
CommonNP type are simple common noun phrases (with possible proper noun modifiers, e.g. "*the Kodak camera*").

We use a Java-written shallow parser from the OpenNLP (http://opennlp.sourceforge.net/) package. Each sentence is tokenized, tagged with part-of-speech, and tagged with noun phrase boundaries. The pattern matching and extraction is straightforward.

TEG mode

TEG (Trainable Extraction Grammars) (Rosenfeld, Feldman et al. 2004) is general-purpose hybrid rule-based and statistical IE system, able to extract entities and relations at the sentence level. It is adapted to any domain by writing a suitable set of rules, and training them using an annotated corpus. The TEG rule language is a straightforward extension of a context-free grammar syntax. A complete set of rules is compiled into a PCFG (Probabilistic Context Free Grammar), which is then trained upon the training corpus.

Some of the nonterminals inside the TEG grammar can be marked as *target concepts*. Wherever such nonterminal occurs in a final parse of a sentence, TEG generates an output label. The target concept rules may specify some of their parts as attributes. Then the concept is considered to be a relation, with the values of the attributes determined by the concept parse. Concepts without attributes are entities.

For the TEG-based instance extractor we utilize the NER ruleset of TEG and an internal training corpus called INC, as described in (Rosenfeld, Feldman et al. 2004). The ruleset defines a grammar with a set of concepts for *Person*, *Location*, and *Organization* entities. In addition, the grammar defines a generic *NounPhrase* concept, which can be used for capturing the entities that do not belong to any of the entity types above.

In order to do the extraction, the patterns generated by the Pattern Learner are converted to the TEG syntax and added to the pre-built NER grammar. This produces a grammar, which is able to extract relations. This grammar is trained upon the automatically labeled positive set from the Pattern Learning. The resulting trained model is applied to the sets of sentences produced by the Sentence Classifier.

4 Experimental Evaluation

In order to evaluate URES, we used five predicates

Acquisition(*BuyerCompany*, *BoughtCompany*),
Merger(*Company1*, *Company2*),
CEO_Of(*Company*, *Name*),
MayorOf(*City*, *Name*),
InventorOf(*InventorName*, *Invention*).

Merger is *symmetric* predicate, in the sense that the order of its attributes does not matter. *Acquisition* is *antisymmetric*, and the other three are tested as *bound* in the first attribute. For the bound predicates, we are only interested in the instances with particular prespecified values of the first attribute.

We test both modes of operation – using shallow parser and using TEG. In the shallow parser mode, the *Invention* attribute of the *InventorOf* predicate is of type *CommonNP*, and all other attributes are of type *ProperName*. In the TEG mode, the "*Company*" attributes are of type *Organization*, the "*Name*" attributes are of type *Person*, the "*City*" attribute is of type *Location*, and the "*Invention*" attribute is of type *NounPhrase*.

We evaluate our system by running it over a large set of sentences, counting the number of extracted instances, and manually checking a random sample of the instances to estimate precision. In order to be able to compare our results with KnowItAll-produced results, we used the set of sentences collected by the KnowItAll's crawler as if they were produced by the Sentence Gatherer.

The set of sentences for the *Acquisition* and *Merger* predicates contained around 900,000 sentences each. For the other three predicates, each of the sentences contained one of the 100 predefined values for the first attribute. The values (100 companies for *CEO_Of*, 100 cities for *MayorOf*, and 100 inventors for *InventorOf*) are entities collected by KnowItAll, half of them are frequent entities (>100,000 hits), and another half are rare (<10,000 hits).

In all of the experiments, we use ten top predicate instances extracted by KnowItAll for the relation seeds needed by the Pattern Learner.

The results of our experiments are summarized in the Table 2. The table displays the number of extracted instances and estimated precision for three different URES setups, and for the KnowItAll manually built patterns. Three results are shown for each setup and each relation – extractions supported by at least one, at least two, and at least three different sentences, respectively.

Several conclusions can be drawn from the results. First, both modes of URES significantly outperform KnowItAll in recall (number of extractions), while maintaining the same level of precision or improving it. This demonstrates utility of our pattern learning component. Second, it is immediately apparent, that using only anchored patterns significantly improves precision of NP Tagger-based URES, though at a high cost in recall. The NP tagger-based URES with anchored patterns performs somewhat worse than

Table 2 - Experimental results.

		Acquisition		*CEO_Of*		*InventorOf*		*MayorOf*		*Merger*	
	support	Count	Prec	Count	Prec	Count	Prec	Count	Prec	Count	Prec
NP Tagger All patterns	≥ 1	10587	0.74	545	0.7	1233	0.84	2815	0.6	25071	0.71
	≥ 2	815	0.87	221	0.92	333	0.92	717	0.74	2981	0.8
	≥ 3	234	0.9	133	0.94	185	0.96	442	0.84	1245	0.88
NP Tagger Anchored patterns	≥ 1	5803	0.84	447	0.8	1035	0.86	2462	0.65	17107	0.8
	≥ 2	465	0.96	186	0.94	284	0.92	652	0.78	2481	0.83
	≥ 3	148	0.98	123	0.96	159	0.96	411	0.88	1084	0.9
TEG All patterns	≥ 1	8926	0.82	618	0.83	2322	0.65	2434	0.85	15002	0.8
	≥ 2	1261	0.94	244	0.94	592	0.85	779	0.93	2932	0.86
	≥ 3	467	0.98	158	0.98	334	0.88	482	0.98	1443	0.9
KnowItAll	≥ 1	2235	0.84	421	0.81	604	0.8	725	0.76	3233	0.82
	≥ 2	257	0.98	190	0.98	168	0.92	308	0.92	352	0.92

TEG-based URES on all predicates except *InventorOf*, as expected. For the *InventorOf*, TEG performs worse, because of overly simplistic implementation of the *NounPhrase* concept inside the TEG grammar – it is defined as a sequence of zero or more adjectives followed by a sequence of nouns. Such definition often leads to only part of a correct invention name being extracted.

5 Conclusions and Future Work

We have presented the URES system for autonomously extracting relations from the Web. URES bypasses the bottleneck created by classic information extraction systems that either relies on manually developed extraction patterns or on manually tagged training corpus. Instead, the system relies upon learning patterns from a large unlabeled set of sentences downloaded from Web.

One of the topics we would like to further explore is the complexity of the patterns that we learn. Currently we use a very simple pattern language that just has 4 types of elements, slots, constants and two types of skips. We want to see if we can achieve higher precision with more complex patterns. In addition we would like to test URES on n-ary predicates, and to extend the system to handle predicates that are allowed to lack some of the attributes.

References

Agichtein, E. and L. Gravano (2000). Snowball: Extracting Relations from Large Plain-Text Collections. Proceedings of the 5th ACM International Conference on Digital Libraries (DL).

Bikel, D. M., S. Miller, et al. (1997). Nymble: a high-performance learning name-finder. Proceedings of ANLP-97**:** 194-201.

Brin, S. (1998). Extracting Patterns and Relations from the World Wide Web. WebDB Workshop, EDBT '98.

Califf, M. E. and R. J. Mooney (1998). Relational Learning of Pattern-Match Rules for Information Extraction. Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing. Menlo Park, CA, AAAI Press**:** 6-11.

Chinchor, N., L. Hirschman, et al. (1994). "Evaluating Message Understanding Systems: An Analysis of the Third Message Understanding Conference (MUC-3)." Computational Linguistics **3**(19): 409-449.

Collins, M. and S. Miller (1998). Semantic Tagging using a Probabilistic Context Free Grammar. Proceedings of the Sixth Workshop on Very Large Corpora.

Etzioni, O., M. Cafarella, et al. (2005). "Unsupervised named-entity extraction from the Web: An experimental study." Artificial Intelligence.

Fisher, D., S. Soderland, et al. (1995). Description of the UMass Systems as Used for MUC-6. 6th Message Understanding Conference**:** 127-140.

Manning, C. and H. Schutze (1999). Foundations of Statistical Natural Language Processing. Cambridge, US, The MIT Press.

McCallum, A., D. Freitag, et al. (2000). Maximum Entropy Markov Models for Information Extraction and Segmentation. Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA**:** 591-598.

Miller, D., R. Schwartz, et al. (1999). Named entity extraction from broadcast news. Proceedings of DARPA Broadcast News Workshop. Herndon, VA.

Miller, G. A. (1995). "WordNet: A lexical database for English." CACM **38**(11): 39-41.

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

Riloff, E. (1996). Automatically Generating Extraction Patterns from Untagged Text. AAAI/IAAI, Vol. 2**:** 1044-1049.

Riloff, E. and R. Jones (1999). Learning Dictionaries for Information Extraction by Multi-level Bootstrapping. Proceedings of the Sixteenth National Conference on Artificial Intelligence, The AAAI Press/MIT Press**:** 1044-1049.

Rosenfeld, B., R. Feldman, et al. (2004). TEG: a hybrid approach to information extraction. CIKM 2004, Arlington, VA.

Soderland, S. (1999). "Learning Information Extraction Rules for Semi-Structured and Free Text." Machine Learning **34**(1-3): 233-272.

Sudo, K., S. Sekine, et al. (2001). Automatic pattern acquisition for Japanese information extraction. Human Language Technology Conference (HTL2001).

Thelen, M. and E. Riloff (2002). A Bootstrapping Method for Learning Semantic Lexicons using Extraction Pattern Contexts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2002).

Yeh, A. and L. Hirschman (2002). "Background and overview for kdd cup 2002 task 1: Information extraction from biomedical articles." KDD Explorarions **4**(2): 87-89.

Zelle, J. M. and R. J. Mooney. (1996). Learning to parse database queries using inductive logic programming. 13th National Conference on Artificial Intelligence (AAAI-96).