Boosting Unsupervised Relation Extraction by Using NER

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

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

Abstract

Web extraction systems attempt to use the immense amount of unlabeled text in the Web in order to create large lists of entities and relations. Unlike traditional IE methods, the Web extraction systems do not label every mention of the target entity or relation, instead focusing on extracting as many different instances as possible while keeping the precision of the resulting list reasonably high. URES is a Web relation extraction system that learns powerful extraction patterns from unlabeled text, using short descriptions of the target relations and their attributes. The performance of URES is further enhanced by classifying its output instances using the properties of the extracted patterns. The features we use for classification and the trained classification model are independent from the target relation, which we demonstrate in a series of experiments. In this paper we show how the introduction of a simple rule based NER can boost the performance of URES on a variety of relations. We also compare the performance of URES to the performance of the state-of-the-art KnowItAll system, and to the performance of its pattern learning component, which uses a simpler and less powerful pattern language than URES.

1 Introduction

Information Extraction (IE) (Riloff 1993; Cowie and Lehnert 1996; Grishman 1996; Grishman 1997; Kushmerick, Weld et al. 1997; Freitag 1998; Freitag and McCallum 1999; Soderland 1999) is the task of extracting factual assertions from text.

Most IE systems rely on knowledge engineering or on machine learning to generate extraction patterns – the mechanism that extracts entities and relation instances from text. In the machine learning approach, a domain expert labels instances of the target relations in a set of documents. The system then learns extraction patterns, which can be applied to new documents automatically.

Both approaches require substantial human effort, particularly when applied to the broad range of documents, entities, and relations on the Web. In order to minimize the manual effort necessary to build Web IE systems, we have designed and implemented URES (Unsupervised Relation Extraction System). URES takes as input the names of the target relations and the types of their arguments. It then uses a large set of unlabeled documents downloaded from the Web in order to learn the extraction patterns.

URES is most closely related to the KnowItAll system developed at University of Washington by Oren Etzioni and colleagues (Etzioni, Cafarella et al. 2005), since both are unsupervised and both leverage relation-independent extraction patterns to automatically generate seeds, which are then fed into a pattern-learning component. KnowItAll is based on the observation that the Web corpus is highly redundant. Thus, its selective, high-precision extraction patterns readily ignore most sentences, and focus on sentences that indicate the presence of relation instances with very high probability.

In contrast, URES is based on the observation that, for many relations, the Web corpus has *limited redundancy*, particularly when one is concerned with less prominent instances of these relations (e.g., the acquisition of Austria Tabak). Thus, URES utilizes a more expressive extraction pattern language, which enables it to extract information from a broader set of sentences. URES relies on a sophisticated mechanism to

Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006), pages 473–481,
Sydney, July 2006.

assess its confidence in each extraction, enabling it to sort extracted instances, thereby improving its recall without sacrificing precision.

Our main contributions are as follows:

- We introduce the first domain-independent system to extract relation instances from the Web with both high precision and high recall.
- We show how to minimize the human effort necessary to deploy URES for an arbitrary set of relations, including automatically generating and labeling positive and negative examples of the relation.
- We show how we can integrate a simple NER component into the classification scheme of URES in order to boost recall between 5-15% for similar precision levels.
- We report on an experimental comparison between URES, URES-NER and the state-of-the-art KnowItAll system, and show that URES can double or even triple the recall achieved by KnowItAll for relatively rare relation instances.

The rest of the paper is organized as follows: Section 2 describes previous work. Section 3 outlines the general design principles of URES, its architecture, and then describes each URES component in detail. Section 4 presents our experimental evaluation. Section 5 contains conclusions and directions for future work.

2 Related Work

The IE systems most similar to URES are based on bootstrap learning: Mutual Bootstrapping (Riloff and Jones 1999), the DIPRE system (Brin 1998), and the Snowball system (Agichtein and Gravano 2000). (Ravichandran and Hovy 2002) also use bootstrapping, and learn simple surface patterns for extracting binary relations from the Web.

Unlike those unsupervised IE systems, URES patterns allow gaps that can be matched by any sequences of tokens. This makes URES patterns much more general, and allows to recognize instances in sentences inaccessible to the simple surface patterns of systems such as (Brin 1998; Riloff and Jones 1999; Ravichandran and Hovy 2002). The greater power of URES requires different and more complex methods for learning, scoring, and filtering of patterns.

Another direction for unsupervised relation learning was taken in (Hasegawa, Sekine et al. 2004; Chen, Ji et al. 2005). These systems use a NER system to identify pairs of entities and then cluster them based on the types of the entities and the words appearing between the entities. Only pairs that appear at least 30 times were considered. The main benefit of this approach is that all relations between two entity types can be discovered simultaneously and there is no need for the user to supply the relations definitions. Such a system could have been used as a preliminary step to URES, however its relatively low precision makes it unfeasible. Unlike URES, the evaluations performed in these papers ignored errors that were introduced by the underlying NER component. The precision reported by these systems (77% breakeven for the COM-COM domain) is inferior to that of URES.

We compared our results directly to two other unsupervised extraction systems, the Snowball (Agichtein and Gravano 2000) and KnowItAll. Snowball 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 fledged Named Entity Recognition system. Snowball achieved fairly low precision figures (30-50%) on relations such as *Merger* and *Acquisition* on the same dataset we used in our experiments.

KnowItAll is a system developed at University of Washington by Oren Etzioni and colleagues (Etzioni, Cafarella et al. 2005). We shall now briefly describe it and its pattern learning component.

Brief description of KnowItAll

KnowItAll uses a set of generic extraction patterns, and automatically instantiates rules by combining those patterns with user supplied relation labels. For example, KnowItAll has patterns for a generic “of” relation:

NP1 <relation> NP2
NP1 's <relation> , NP2
NP2 , <relation> of NP1

where NP1 and NP2 are simple noun phrases that extract values of attribute1 and attribute2 of a relation, and <relation> is a user-supplied string associated with the relation. The rules may also constrain NP1 and NP2 to be proper nouns.

The rules have alternating context strings (exact string match) and extraction slots (typically an NP or head of an NP). Each rule has an associated query used to automatically find candidate sentences from a Web search engine.

KnowItAll also includes mechanisms to control the amount of search, to merge redundant extractions, and to assign a probability to each extraction based on frequency of extraction or on Web statistics (Downey, Etzioni et al. 2004).

KnowItAll-PL. While those generic rules lead to high precision extraction, they tend to have low recall, due to the wide variety of contexts describing a relation. KnowItAll includes a simple pattern learning scheme (KnowItAll-PL) that builds on the generic extraction mechanism (KnowItAll-baseline). Like URES, this is a self-supervised method that bootstraps from seeds that are automatically extracted by the baseline system.

KnowItAll-PL creates a set of positive training sentences by downloading sentences that contain both argument values of a seed tuple and also the relation label. Negative training is created by downloading sentences with only one of the seed argument values, and considering a nearby NP as the other argument value. This does not guarantee that the negative example will actually be false, but works well in practice.

Rule induction tabulates the occurrence of context tokens surrounding the argument values of the positive training sentences. Each candidate extraction pattern has a left context of zero to k tokens immediately to the left of the first argument, a middle context of all tokens between the two arguments, and a right context of zero to k tokens immediately to the right of the second argument. A pattern can be generalized by dropping the furthest terms from the left or right context. KnowItAll-PL retains the most general version of each pattern that has training frequency over a threshold and training precision over a threshold.

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 (composed of the relation's schema and a few keywords that enable gathering relevant sentences). For example, this is the description of the acquisition relation:

Acquisition(ProperNP, ProperNP) ordered
keywords={"acquired" "acquisition"}

The word ordered indicates that Acquisition is not a symmetric relation and the order of its arguments matters. The ProperNP tokens indicate the types of the attributes. In the regular mode, there are only two possible attribute types – ProperNP and CommonNP, meaning proper and common noun phrases, respectively. When using the NER Filter component described in the section 4.1 we allow further subtypes of ProperNP, and the predicate definition becomes:

acquisition(Company, Company) …

The keywords are used for gathering sentences from the Web and for instantiating the generic patterns for seeds generation. Additional keywords (such as “acquire”, “purchased”, “hostile takeover”, etc), which can be used for gathering more sentences, are added automatically by using WordNet [18].

URES consists of several largely independent components; their layout is shown on the Figure 1. The Sentence Gatherer generates (e.g., downloads from the Web) a large set of sentences that may contain target instances. The Seeds Generator, which is essentially equal to the KnowItAll-baseline system, uses a small set of generic patterns instantiated with the predicate keywords to extract a small set of high-confidence instances of the target relations. The Pattern Learner uses the seeds to learn likely patterns of relation occurrences. Then, the Instance Extractor uses the patterns to extracts the instances from the sentences. Those instances can be filtered by a NER Filter, which is an optional part of the system. Finally, the Classifier assigns the confidence score to each extraction.

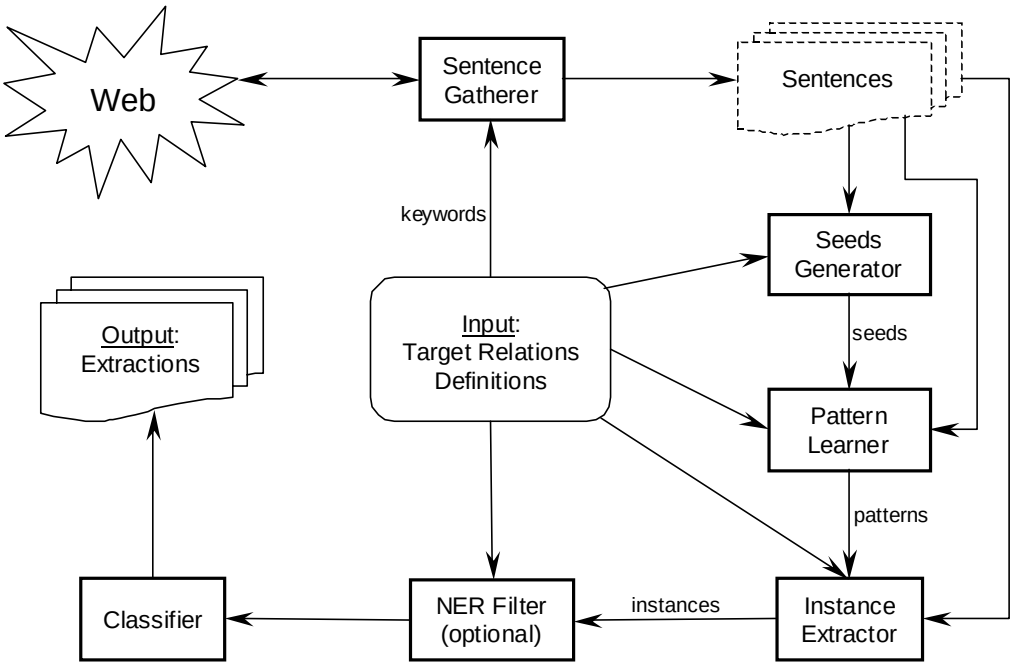

Figure 1. The architecture of URES

3.1 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 instances in our experiments) of known instances for each target relation. Our system assumes that the seeds are a part of the target relation definition. However, the set of seeds need not be created manually. Instead, the seeds can be taken automatically 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: we used two generic patterns instantiated with the relation name and keywords. Those patterns have a relatively high precision (although low recall), and the top-confidence results, which are the ones extracted many times from different sentences, have close to 100% probability of being correct.

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 patterns are learned. Finally, 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 sentences with known false instances of the predicate similarly marked (with *<AttrN>* substituted for attributes). The negative set is used by the pattern learner during the scoring and filtering step, to filter out the patterns that are overly general. We generate the negative set from the sentences in the positive set by

changing the assignment of one or both attributes to other suitable entities in the sentence. In the shallow parser based mode of operation, any suitable noun phrase can be assigned to an attribute.

GENERATING THE PATTERNS

The patterns for the 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 pseudocode shows the process of generating the patterns for the 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. In the shallow parser based mode, there are only two different entity types – *ProperNP* and *CommonNP*, standing for proper and common noun phrases.

The *Generalize*(s_1, s_2) function takes two sentences 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. The exact costs of matching elements are not important as long as their relative order is maintained. 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 dynamic programming-based search, the following match will be found:

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. Assuming that "X" belongs to the same type as at least one of the attributes while the other tokens are not entities, the match will be converted to the pattern

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

3.2 Classifying the Extractions

The goal of the final classification stage is to filter the list of all extracted instances, keeping the correct extractions and removing mistakes that would always occur regardless of the quality of the patterns. It is of course impossible to know which extractions are correct, but there exist properties of patterns and pattern matches that increase or decrease the confidence in the extractions that they produce. Thus, instead of a binary classifier, we seek a real-valued confidence function c, mapping the set of extracted instances into the [0, 1] segment.

Since confidence value depends on the properties of particular sentences and patterns, it is more properly defined over the set of single pattern matches. Then, the overall confidence of an instance is the maximum of the confidence values of the matches that produce the instance.

Assume that an instance E was extracted from a match of a pattern P at a sentence S.

The following set of binary features may influence the confidence c(E, P, S):

f1(E, P, S) = 1, if the number of sentences producing E is greater than one.

f2(E, P, S) = 1, if the number of sentences producing E is greater than two.

f3(E, P, S) = 1, if at least one slot of the pattern P is adjacent to a non-stop-word token.

f4(E, P, S) = 1, if both slots of the pattern P are adjacent to non-stop-word tokens.

f5…f9(E, P, S) = 1, if the number of nonstop words in P is 0 (f5), 1 or greater (f6), 2 or greater (f7), 3 or greater (f8), and 4 or greater (f9).

f10…f15(E, P, S) = 1, if the number of words between the slots of the match M that were matched to skips of the pattern P is 0 (f10), 1 or less (f11), 2 or less (f12) , 3 or less(f13), 5 or less (f14), and 10 or less (f15).

Utilizing the NER

In the URES-NER version the entities of each candidate instance are passed through a simple rule-based NER filter, which attaches a score ("yes", "maybe", or "no") to the argument(s) and optionally fixes the arguments boundaries. The NER is capable of identifying entities of type PERSON and COMPANY (and can be extended to identify additional types).

The scores mean:

"yes" – the argument is of the correct entity type.

"no" – the argument is not of the right entity type, and hence the candidate instance should be removed.

"maybe" – the argument type is uncertain, can be either correct or no.

If "no" is returned for one of the arguments, the instance is removed. Otherwise, an additional binary feature is added to the instance's vector:

f16 = 1 iff the score for both arguments is "yes".

For bound predicates, only the second argument is analyzed, naturally.

As can be seen, the set of features above is small, and is not specific to any particular predicate. This allows us to train a model using a small amount of labeled data for one predicate, and then use the model for all other predicates:

Training: The patterns for a single model predicate are run over a relatively small set of sentences (3,000-10,000 sentences in our experiments), producing a set of extractions (between 150-300 extractions in our experiments).

The extractions are manually labeled according to whether they are correct or not.

For each pattern match Mk = (Ek, Pk, Sk), the value of the feature vector fk = (f1(Mk), …, f15(Mk)) is calculated, and the label Lk = ±1 is set according to whether the extraction Ek is correct or no.

A regression model estimating the function L(f) is built from the training data {(fk, Lk)}. For our classifier we used the BBR (Genkin, Lewis et al. 2004), but other models, such as SVM or NaiveBayes are of course also possible.

Confidence estimation: For each pattern match M, its score L(f(M)) is calculated by the trained regression model. Note that we do not threshold the value of L, instead using the raw probability value between zero and one.

The final confidence estimates c(E) for the extraction E is set to the maximum of L(f(M)) over all matches M that produced E.

4 Experimental Evaluation

Our experiments aim to answer three questions:

1. Can we train URES's classifier once, and then use the results on all other relations?
2. What boost will we get by introducing a simple NER into the classification scheme of URES?
3. How does URES's performance compare with KnowItAll and KnowItAll-PL?

Our experiments utilized five relations:

Acquisition(BuyerCompany,AcquiredCompany),
Merger(Company1, Company2),
CEO_Of(Company, Person),
MayorOf(City, Person),
InventorOf(Person, Invention).

Merger is a 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. The Invention attribute of the InventorOf predicate is of type CommonNP. All other attributes are of type ProperName.

The data for the experiments were collected by the KnowItAll crawler. The data for the Acquisition and Merger predicates consist of about 900,000 sentences for each of the two predicates, where each sentence contains at least one predicate keyword. The data for the bounded predicates consist of sentences that contain a predicate keyword and one of a hundred values of the first (bound) attribute. Half of the hundred are frequent entities (>100,000 search engine hits), and another half are rare (<10,000 hits).

The pattern learning for each of the predicates was performed using the whole corpus of sentences for the predicate. For testing the precision of each of the predicates in each of the systems we manually evaluated sets of 200 instances that were randomly selected out of the full set of instances extracted from the whole corpus.

In the first experiment, we test the performance of the classification component using different predicates for building the model. In the second experiment we evaluate the full system over the whole dataset.

4.1 Cross-Predicate Classification Performance

In this experiment we test whether the choice of the model predicate for training the classifier is significant.

The pattern learning for each of the predicates was performed using the whole corpus of sentences for the predicate. For testing we used a small random selection of sentences, run the Instance Extractor over them, and manually evaluated each extracted instance. The results of the evaluation for Acquisition, CEO_Of, and Merger are summarized in Figure 2. As can be seen, using any of the predicates as the model produces similar results. The graphs for the other two predicates are similar. We have used only the first 15 features, as the NER-based feature (f_{16}) is predicate-dependent.

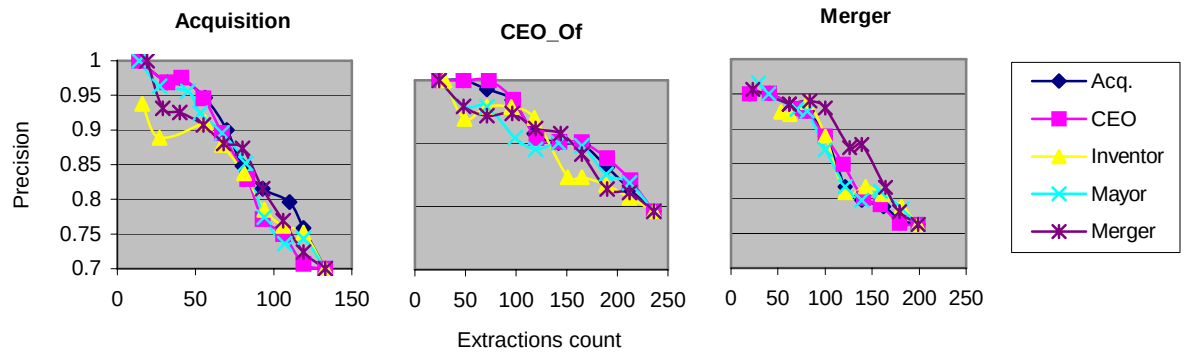

Figure 2. Cross-predicate classification performance results. Each graph shows the five precision-recall curves produced by using the five different model predicates. As can be seen, the curves on each graph are very similar.

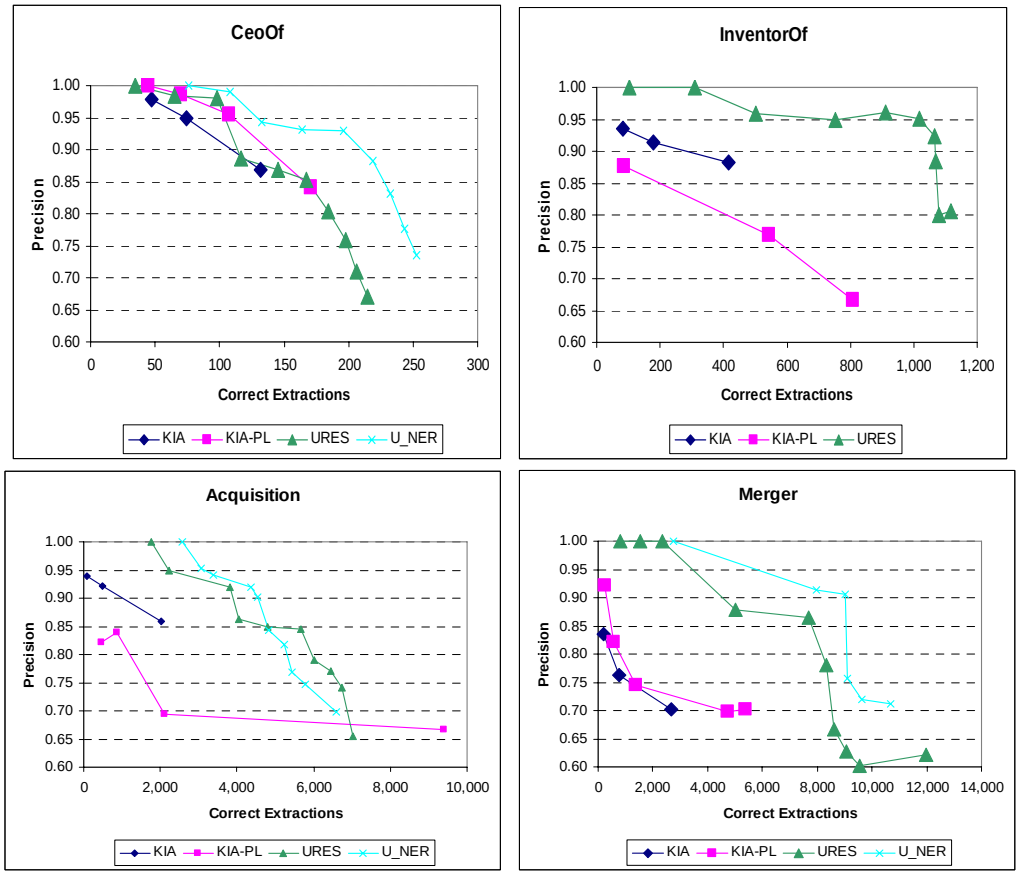

Figure 3. Comparision between URES, URES-NER, KnowItAll-baseline, and KnowItAll-PL.

4.2 Performance of the whole system

In this experiment we compare the performance of URES with classification to the performance of KnowItAll. To carry out the experiments, we used extraction data kindly provided by the KnowItAll group. They provided us with the extractions obtained by the KnowItAll system and by its pattern learning component (KnowItAll-PL). Both are sketched in Section 2.1 and are described in detail in (Etzioni, Cafarella et al. 2005).

In this experiment we used Acquisition as the model predicate for testing all other predicates except itself. For testing Acquisition we used CEO_Of as the model predicate. The results are summarized in the five graphs in the Figure 3.

For three relations (Acquisition, Merger, and InventorOf) URES clearly outperforms KnowItAll. Yet for the other two (CEO_Of and MayorOf), the simpler method of KnowItAll-PL or even the KnowItAll-baseline do as well as URES. Close inspection reveals that the key difference is the amount of redundancy of instances of those relations in the data. Instances of CEO_Of and MayorOf are mentioned frequently in a wide variety of sentences whereas instances of the other relations are relatively infrequent.

KnowItAll extraction works well when redundancy is high and most instances have a good chance of appearing in simple forms that KnowItAll is able to recognize. The additional machinery in URES is necessary when redundancy is low. Specifically, URES is more effective in identifying low-frequency instances, due to its more expressive rule representation, and its classifier that inhibits those rules from overgeneralizing.

In the same graphs we can see that URES-NER outperforms URES by 5-15% in recall for similar precision levels. We can also see that for Person-based predicates the improvement is much more pronounced, because Person is a much simpler entity to recognize. Since in the InventorOf predicate the 2nd attribute is of type CommonNP, the NER component adds no value and URES-NER and URES results are identical for this predicate.

5 Conclusions

We have presented the URES system for autonomously extracting relations from the Web. We showed how to improve the precision of the system by classifying the extracted instances using the properties of the patterns and sentences that generated the instances and how to utilize a simple NER component. The cross-predicate tests showed that classifier that performs well for all relations can be built using a small amount of labeled data for any particular relation. We performed an experimental comparison between URES, URES-NER and the state-of-the-art KnowItAll system, and showed that URES can double or even triple the recall achieved by KnowItAll for relatively rare relation instances, and get an additional 5-15% boost in recall by utilizing a simple NER. In particular we have shown that URES is more effective in identifying low-frequency instances, due to its more expressive rule representation, and its classifier (augmented by NER) that inhibits those rules from overgeneralizing.

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).

Brin, S. (1998). Extracting Patterns and Relations from the World Wide Web. WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98, Valencia, Spain.

Chen, J., D. Ji, et al. (2005). Unsupervised Feature Selection for Relation Extraction IJCNLP-05, Jeju Island, Korea.

Cowie, J. and W. Lehnert (1996). "Information Extraction." Communications of the Association of Computing Machinery **39**(1): 80-91.

Downey, D., O. Etzioni, et al. (2004). Learning Text Patterns for Web Information Extraction and Assessment (Extended Version). Technical Report UW-CSE-04-05-01.

Etzioni, O., M. Cafarella, et al. (2005). "Unsupervised named-entity extraction from the Web: An experimental study." Artificial Intelligence **165**(1): 91-134.

Freitag, D. (1998). Machine Learning for Information Extraction in Informal Domains. Computer Science Department. Pittsburgh, PA, Carnegie Mellon University**:** 188.

Freitag, D. and A. K. McCallum (1999). Information extraction with HMMs and shrinkage. Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction.

Genkin, A., D. D. Lewis, et al. (2004). Large-Scale Bayesian Logistic Regression for Text Categorization. New Brunswick, NJ, DIMACS**:** 1-41.

Grishman, R. (1996). The role of syntax in Information Extraction. Advances in Text Processing: Tipster Program Phase II, Morgan Kaufmann.

Grishman, R. (1997). Information Extraction: Techniques and Challenges. SCIE**:** 10-27.

Hasegawa, T., S. Sekine, et al. (2004). Discovering Relations among Named Entities from Large Corpora. ACL 2004.

Kushmerick, N., D. S. Weld, et al. (1997). Wrapper Induction for Information Extraction. IJCAI-97**:** 729-737.

Ravichandran, D. and E. Hovy (2002). Learning Surface Text Patterns for a Question Answering System. 40th ACL Conference.

Riloff, E. (1993). Automatically Constructing a Dictionary for Information Extraction Tasks. AAAI-93.

Riloff, E. and R. Jones (1999). Learning Dictionaries for Information Extraction by Multi-level Boot-strapping. AAAI-99.

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