Knowl Inf Syst (2008) 17:17–33
DOI 10.1007/s10115-007-0110-6

REGULAR PAPER

Self-supervised relation extraction from the Web

Benjamin Rozenfeld · Ronen Feldman

Received: 17 January 2007 / Revised: 5 July 2007 / Accepted: 8 September 2007 /
Published online: 20 November 2007

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 Information Extraction 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. SRES is a self-supervised Web relation extraction system that learns powerful extraction patterns from unlabeled text, using short descriptions of the target relations and their attributes. SRES automatically generates the training data needed for its pattern-learning component. The performance of SRES is further enhanced by classifying its output instances using the properties of the instances and the 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. We also compare the performance of SRES to the performance of the state-of-the-art KnowItAll system, and to the performance of its pattern learning component, which learns simpler pattern language than SRES.

Keywords Web extraction · Text mining · Pattern learning · Unsupervised learning · Relationship extraction

1 Introduction

Information Extraction (IE) [5,8–10,12,13,15,16,19,21,22] 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

B. Rozenfeld · R. Feldman (✉)
Information Systems, HU School of Business Administration, Hebrew University, Jerusalem, Israel
e-mail: Ronen.Feldman@huji.ac.il

B. Rozenfeld
e-mail: grurgrur@gmail.com

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 Self-Supervised Relation Extraction System (SRES). SRES 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.

SRES is most closely related to the KnowItAll system developed at University of Washington by Oren Etzioni and colleagues [7], because both are self-supervised 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 the sentences that indicate the presence of relation instances with high probability.

In contrast, SRES 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. Consequently, SRES utilizes a more expressive extraction pattern language, which enables it to extract information from a broader set of sentences. SRES relies on a sophisticated mechanism to 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 a novel domain-independent system to extract relation instances from the Web with both high precision and relatively high recall.
- We show how to minimize the human effort necessary to deploy SRES 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 Named-Entity Recognition (NER) component into the classification scheme of SRES in order to boost recall between 5 and 15% for similar precision levels.
- We provide an estimation of the true recall of SRES for unbounded relations such as merger and acquisition.
- We report on an experimental comparison between SRES, SRES-NER and the state-of-the-art KnowItAll system, and show that SRES can double or even triple the recall achieved by KnowItAll for relatively rare relation instances.

The rest of the paper is organized as follows: Sect. 2 describes previous work. Section 3 outlines the general design principles of SRES, its architecture, and then describes each SRES component in detail. Section 4 describes our extraction classification schema and Sect. 5 presents our experimental evaluation. Section 6 contains conclusions and directions for future work.

2 Related work

The IE systems most similar to SRES are based on bootstrap learning: Mutual Bootstrapping [20], the DIPRE system [2], and the Snowball system [1]. Ravichandran and Hovy [18] also use bootstrapping and learn simple surface patterns for extracting binary relations from the Web.

Unlike those unsupervised IE systems, SRES surface patterns allow gaps that can be matched by any sequences of tokens. This makes SRES patterns more general, and allows to recognize relation instances in sentences inaccessible to the simpler surface patterns of systems such as in [2,18,20]. On the other hand, we cannot use even more sophisticated patterns that include complex constraints utilized by fully supervised systems such as in [4,21], because this would lead to severe overfitting problems as all our positive and negative examples are generated automatically.

Another direction for unsupervised relation learning was taken in [3,14]. These systems use an NER system to identify frequently co-occuring pairs of entities and then cluster them based on the types of the entities and the words appearing between the entities. The main benefit of this approach is that all relations between the two entity types can be discovered simultaneously, and there is no need for the user to supply the relations, definitions. Such systems can be used as a preliminary step to SRES if their accuracy reaches sufficiently high level.

We compared our results directly to two other self-supervised extraction systems, the Snowball [1] 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 on relations such as *Merger* and *Acquisition* on the same dataset we used in our experiments. Since we are primarily interested in high-precision results, we did not do further comparisons with SnowBall.

KnowItAll is a system developed at University of Washington by Oren Etzioni and colleagues [7]. We shall now briefly describe it and its pattern learning component.

2.1 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 the 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 attribute 2 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. Similar generic patterns are given for "direct-object" relations (such as acquisition) or symmetric relations (such as merger).

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 quantum of search, to merge redundant extractions, and to assign a probability to each extraction based on frequency of extraction or on Web statistics [6].

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 SRES, 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 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 SRES

The goal of SRES 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—it ProperNP and *CommonNP*, meaning proper and common noun phrases, respectively. When using the NER Filter component described in the Sect. 4.1 we allow further subtypes of *ProperNP*, and the predicate definition becomes:

$$acquisition(Company,\ Company)\ldots$$

The keywords are used in three ways: for gathering sentences from the Web, for instantiating the generic patterns for seeds generation, and for filtering out irrelevant patterns. For the third of these tasks we also utilize additional keywords (such as "acquire", "purchased", "hostile takeover", etc.), which are added automatically using WordNet [17].

SRES consists of several largely independent components; their layout is shown on the Fig. 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 an NER Filter, which is an optional part of the system. Finally, the Classifier assigns the confidence score to each extraction.

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

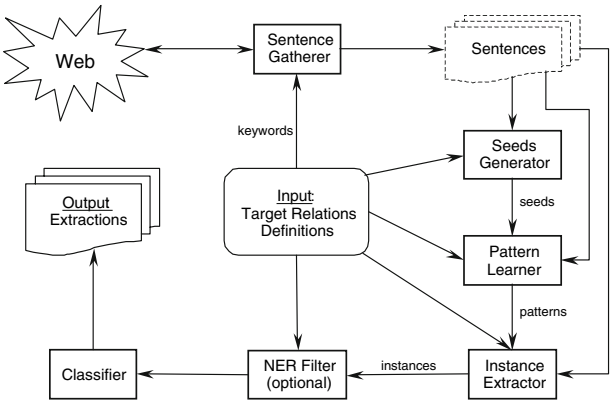

Fig. 1 The architecture of SRES

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. These seeds are generated automatically by the generic patterns (described in Sect. 2.1) instantiated with the relation name and keywords. Those patterns have a relatively high precision (although low recall), and the top-scoring results, which are the ones extracted the highest number of 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 these sets the patterns are learned. Finally, the patterns are post-processed and filtered. We shall now describe the steps in detail.

3.1.1 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

$$The\ Antitrust\ Division \ldots \ldots of <Attr1>'s\ proposed\ acquisition\ of <Attr2>.$$

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). 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. Continuing the example above, the

following sentences will be included in the negative set:

$$<Attr1> of\ the\ <Attr2>\ evaluated\ the\ likely\ \ldots$$
$$<Attr2> of\ the\ U.S.\ \ldots\ \ldots acquisition\ of\ <Attr1>$$
etc.

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 latter case into the negative set of the predicate.

The following pseudocode shows the process of generating the positive and negative sets:

Let S be the set of gathered sentences.
For each predicate P
For each $s \in S$
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 $e_2 \neq e_1$, and
$Type(e_1) =$ type of the first attribute of P, and
$Type(e_2) =$ type of the second attribute 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).
Else If $e_1 = A_2$ and $e_2 = A_1$ and *Symmetric*(P)
Add s' to the *PositiveSet*(P).
Else
Add s' to the *NegativeSet*(P).
The type of an entity—*Type(e)*—is either "*CommonNP*" or "*ProperNP*".

Some of the automatically generated positive or negative sentence may be errors. A positive sentence may be a chance occurrence of the attributes of a true instance in the same sentence, whereas a negative sentence may be generated from a sentence that contained more than one instance of the target predicate, in which case the sentence should not be labeled as negative. However, the number of such mistakes is sufficiently small, and the automatically generated positive and negative sets can be successfully used for pattern learning, as our evaluation demonstrates.

3.1.2 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 *extractable entities*—the entities of the types equal to the types of the predicate attributes. In the shallow parser-based mode, there are only two possible extractable entity types—it ProperNP and *CommonNP*.

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 significant. We use the following numbers: two identical elements match at cost 1, and a token matches an empty space at cost 10. 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: at total

Toward		(cost 10)
	Earlier	(cost 10)
this	*this*	(cost 1)
end		(cost 10)
	Year	(cost 10)
,	,	(cost 1)
<Attr 1>	*<Attr 1>*	(cost 1)
in July		(cost 20)
acquired	*acquired*	(cost 1)
<Attr2>	*<Attr2>*	(cost 1)
	from	(cost 10)
	X	(cost 10)

cost = 85. Assuming that "*X*" is an extractable entity, whereas the other tokens are not entities, the match will be converted to the pattern

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

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.

3.1.3 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, too general, 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 punctuation 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. In particular, because of this we cannot remove stopwords prior to generalization. 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. (All words at depth 1 are insufficient, whereas the set of words of depth 3 is too broad.) 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 the keyword "*acquisition*" via synonym and derivation links.

The filtered patterns are then scored by their performance on the positive and negative sets. The scoring formula reflects the following heuristic: it needs to rise monotonically with the number of positive sentences it matches, but drops fast with the number of negative sentences it matches. Thus, the score of a pattern in our system is the number of positive sentences it matches, divided by the square of the number of negative sentences it matches plus 1:

$$Score(Pattern) = \frac{|\{S \in PositiveSet : Pattern \text{ matches } S\}|}{(|\{S \in PositiveSet : Pattern \text{ matches } S\}| + 1)^2}$$

We tested several other formulas that respected the heuristic above, and chose the best-performing, although the difference in performance is small.

When the scores are known, a threshold is applied to the set of patterns, and all patterns scoring less than the threshold (currently, it is set to 6) are discarded. Finally, if the number of patterns is still too large, the system keeps the 300 top-scoring patterns and discards the rest.

3.1.4 Scalability issues

During pattern creation, the pattern learner calls the *Generalize*() function once for each pair of positive sentences. The complexity of the Generalize function, which is based on dynamical programming, is quadratic in the sentence length. Thus, the complexity of pattern creation is $O(K^2L^2)$, where K is the number of sentences in the *PositiveSet*, and L is the maximal length of sentence measured in tokens. For Web-scale collections, the positive sets of predicates may grow very large. However, useful patterns can be created from any subset of such large positive set, because useful patterns must be sufficiently general to match many positive sentences. In our experiments we picked a random subset of 1,000 positive sentences for each of the predicates. Increasing this number to 2,000 did not significantly change the final results.

The other components of the system are linear in the data size and thus do not present scalability problems.

3.2 Instance extractor

The Instance Extractor applies the patterns generated by the Pattern Learner to a large body of sentences. In order to be able to match the slots of the patterns, the Instance Extractor utilizes an external shallow parser from the OpenNLP package (http://opennlp.sourceforge.net/), which is able to find all proper and common noun phrases in sentences. Those phrases can subsequently be matched to the slots of the patterns. In other respects, the pattern matching and extraction process is straightforward.

3.3 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 properties may influence the confidence $c(E, P, S)$:

- The total number of different sentences, from which the instance E was extracted. Instances extracted from several sentences have much higher confidence.
- Statistics on the pattern P gathered during pattern learning—the number of matched positive and negative sentences.
- Information on whether the slots of the pattern P are adjacent to non-skips. The patterns with such "anchored" slots usually have better precision.
- The total number of non-stop-word tokens that the pattern P contains. The patterns with greater number are more specific and usually more precise.
- Information on whether the sentence S contains entities of the relation attribute's type, between the slots of the match, and outside the match.
- The number of words in the sentence S that were matched to the skips in P that appear between the slots of P. If this number is big, the pattern is less binding, and the confidence is lower.

During the experiments, it turned out that the pattern statistics and the skipped entity information did not lead to any improvement. Other properties were useful, and were turned into the following set of binary features:

$f_1(E, P, S) = 1$, if the number of sentences producing E is ≥ 0.

$f_2(E, P, S) = 1$, if the number of sentences producing E is ≥ 2.

$f_3(E, P, S) = 1$, if at least one slot of the pattern P is adjacent to a non-stop-word token.

$f_4(E, P, S) = 1$, if both slots of the pattern P are adjacent to non-stop-word tokens.

$f_5(E, P, S) = 1$, if the number of non-stop words in P is 0 (f_5), 1 or greater (f_6), 2 or greater (f_7), 3 or greater (f_8), and 4 or greater (f_9).

$f_{10} \ldots f_{15}(E, P, S) = 1$, if the number of words between the slots of the match M that were matched to skips of the pattern P is 0 (f_{10}), $\leq$1(f_{11}), $\leq$2(f_{12}), $\leq$3(f_{13}), $\leq$5(f_{14}), and $\leq$10(f_{15}).

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 $M_k = (E_k, P_k, S_k)$, the value of the feature vector $\boldsymbol{f}_k = (f_1(M_k), \ldots, f_{15}(M_k))$ is calculated, and the label $L_k \in \{1, 0\}$ is set according to whether the extraction E_k is correct or not.

A regression model estimating the function $L(\boldsymbol{f})$ is built from the training data $\{(\boldsymbol{f}_k, L_k)\}$. For our classifier we used the BBR [11], but other models, such as SVM or NaiveBayes are of course also possible.

Confidence estimation: For each pattern match M, its score $L(\boldsymbol{f}(M))$ 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 $L(\boldsymbol{f}(M))$ over all matches M that produced E.

3.4 NER filter

In the SRES–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 filter is capable of identifying entities of type PERSON and COMPANY. 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:

$f_{16} = 1$ iff the score for both arguments is "yes".

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

The NER filter is implemented as a Perl script that checks whether a character string conforms to a set of simple regular expression patterns, and whether it appears inside lists of known entities. The patterns represent simple regularities in the internal structure of the entity types. For example, the patterns for PERSON include:

Person = KnownFirstName [Initial] LastName
Person = Honorific [FirstName] [Initial] LastName
Honorific = ("Mr" | "Ms" | "Dr" | . . .) ["."]
Initial = CapitalLetter ["."]
KnownFirstName = *member of KnownPersonalNamesList*
FirstName = CapitalizedWord
LastName = CapitalizedWord
LastName = CapitalizedWord ["–"CapitalizedWord]
LastName = ("o" | "de" | . . .) "'"CapitalizedWord

while the patterns for COMPANY include:

Company = KnownCompanyName
Company = CompanyName CompanyDesignator
Company = CompanyName FrequentCompanySfx
KnownCompanyName = *member of KnownCompaniesList*
CompanyName = CapitalizedWord +
CompanyDesignator = "inc" | "corp" | "co" | . . .
FrequentCompanySfx = "systems" | "software" | . . .
. . .

The filter works in the following way: it receives a sentence with a labeled candidate entity of a specified entity type (which in the current version can be either Person or Company). It then applies all of the regular expression patterns to the labeled text and to its enclosing context. If a boundary is incorrectly placed according to the patterns, it is fixed. Then, the following result is returned:

"yes", if some pattern of the right entity type matched the candidate entity, while there were no matches for patterns of other entity types.

"no", if no pattern of the right entity type matched the candidate entity, while there was at least one match of a patterns of other entity types.

"maybe", otherwise, that is either if there were no matches at all, or if both correct and incorrect entity types matched.

4 Experimental evaluation

Our experiments aim to answer three questions:

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

The 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 *ProperNP*.

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).

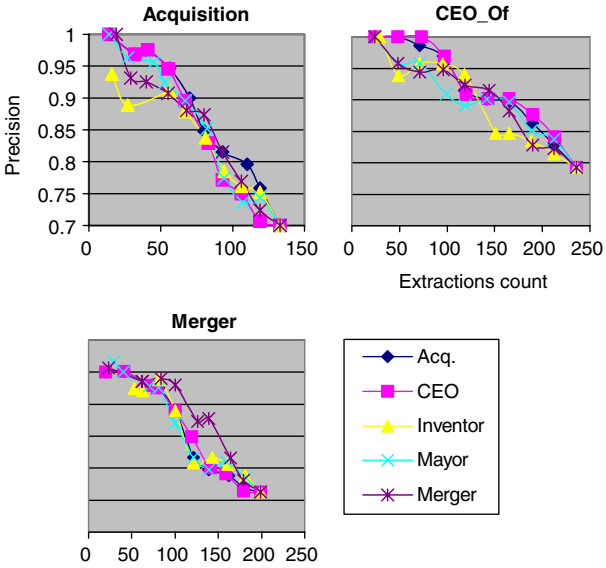

Fig. 2 Cross-predicate classification performance results. Each graph shows the five precision-recall curves produced by using the five different model predicates. As can be seen, the curves on each graph are very similar

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. The manual evaluation was done based on a set of strict evaluation rules that were developed as joint work of the teams of KnowItAll and SRES. Based on these rules all of the dubious instances were considered "mistakes". For instance, instances that had any of the following problems were considered mistakes: extra/missing words in entities, mergers/acquisitions of non-companies (such as schools or political parties), locations included in company names, etc.

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 Fig. 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.

4.2 Performance of the whole system

In this experiment we compare the performance of SRES with classification to the performance of KnowItAll. To carry out the experiments, we used extraction data kindly provided

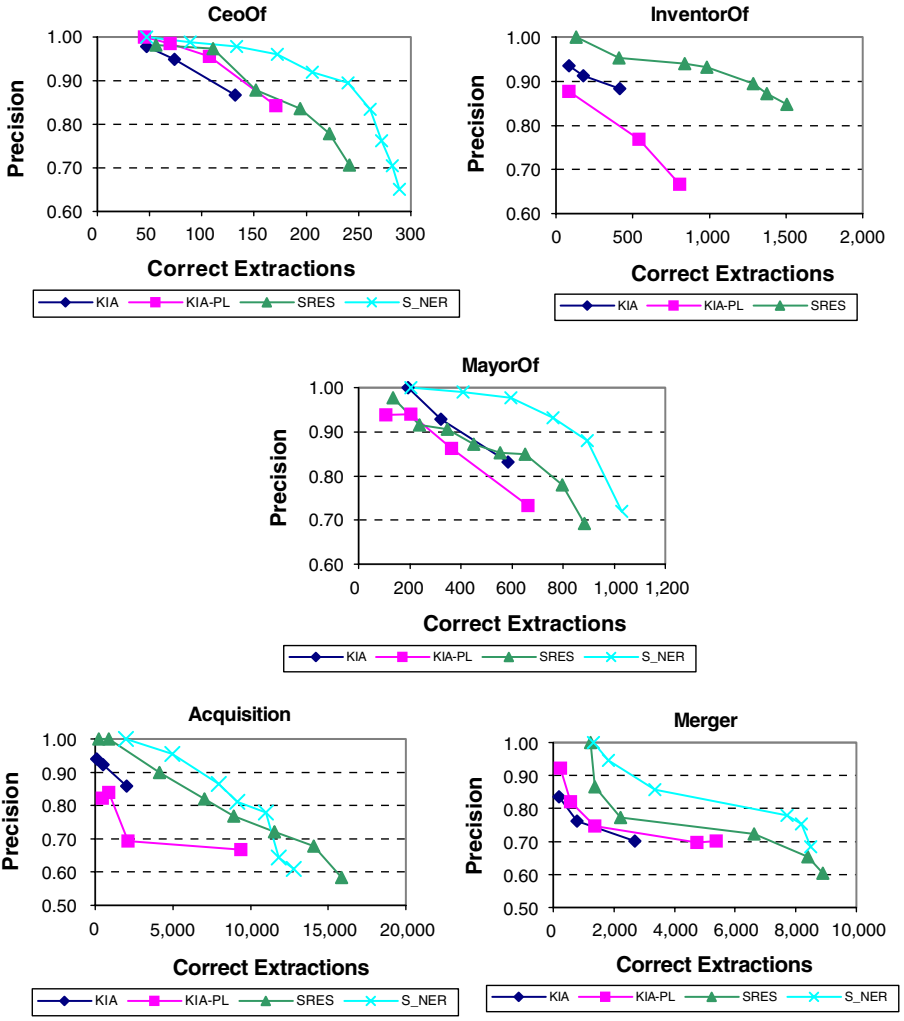

Fig. 3 Comparison between SRES, KnowItAll-baseline, and KnowItAll-PL

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 Sect. 2.1 and are described in detail in [7].

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 Fig. 3.

For three relations (*Acquisition*, *Merger*, and *InventorOf*) SRES clearly outperforms KnowItAll. Yet for the other two (*CEO_Of* and *MayorOf*), the simpler method of KnowItAll-PL or even the KnowItAll-baseline does as well as SRES. 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.

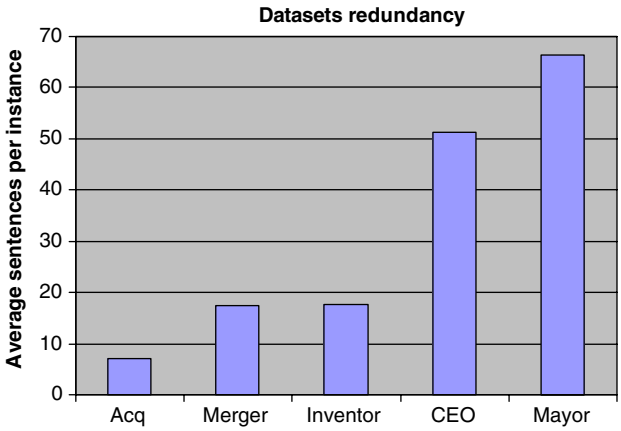

Fig. 4 Data Sets Redundancy

The graph in Fig. 4 shows the difference in the amount of redundancy between the relations, in terms of the average number of sentences that mention each instance of a particular relation. While the numbers are only crude estimates, calculated by counting the sentences that contain both instance attributes, they still clearly demonstrate the qualitative difference between the two sets of relations.

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 SRES is necessary when redundancy is low. Specifically, SRES 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 SRES-NER outperforms SRES 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 second attribute is of type CommonNP, the NER component adds no value and SRES-NER and SRES results are identical for this predicate.

4.3 True recall estimate

In this experiment we attempt to estimate the true recall of the system. It is impossible to manually annotate all of the relation instances because of the huge size of the input corpus. Thus, indirect methods must be used. We used a large list of known acquisition and merger instances (that occurred between 1/1/2004 and 31/12/2005) taken from the paid service subscription SBC Platinum. For each of the instances in this list we identified all sentences in the input corpus that contained both instance attributes and assumed that all such sentences are true instances of the corresponding relation. This is of course an overestimate because in some cases the appearance of both attributes of a true relation instance is just a chance occurrence and does not constitute a true mention of the relation. Thus, our estimates of the true recall are pessimistic, and the actual recall is higher.

For all of the known true instances that appeared in the corpus we count the number of instances that were found by SRES. The fraction of found instances versus all instances gives our recall estimate. In Fig. 5 we present the results for the *Acquisition* and *Merger* relations. We show the recall values at several levels of *redundancy*. The redundancy of an instance in

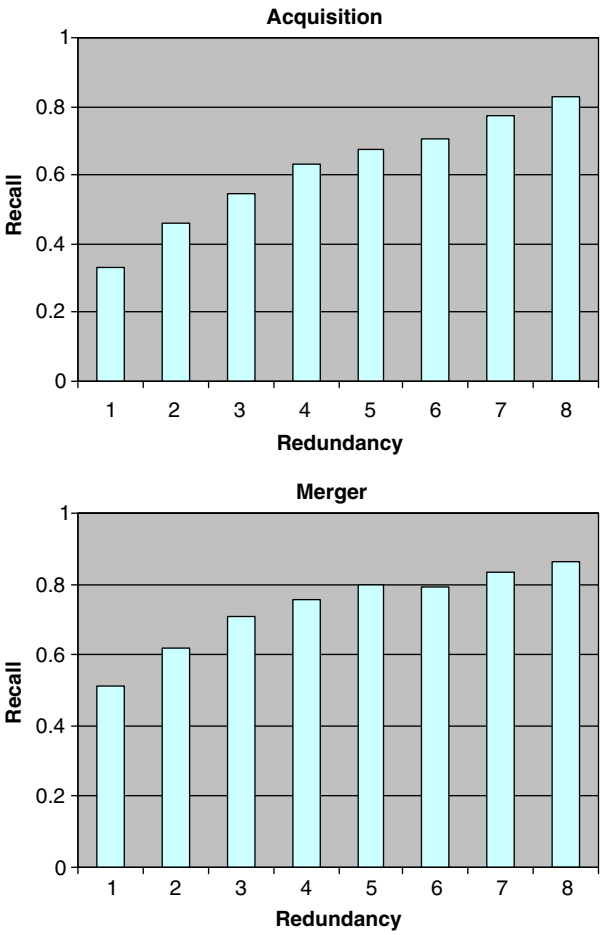

Fig. 5 True recall estimates

a corpus is the number of sentences in which both attributes of the instance appear. As can be seen from the bar charts in the Fig. 5, SRES identifies at least 30% of instances even when there is no redundancy.

5 Conclusions

We have presented the SRES 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 a 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 SRES, SRES-NER, and the state-of-the-art KnowItAll system, and showed that SRES 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 SRES 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.

Acknowledgments Some of the data sets were provided by the KnowItAll project at the University of Washington's Turing Center. We thank Oren Etzioni and Stephen Soderland for helpful discussions.

References

1. Agichtein E, Gravano L (2000) Snowball: extracting relations from large plain-text collections. In: Proceedings of the 5th ACM international conference on digital libraries (DL)
2. Brin S (1998) Extracting patterns and relations from the World Wide Web. In: WebDB workshop at 6th international conference on extending database technology, EDBT'98, Valencia
3. Chen J, Ji D et al (2005) Unsupervised feature selection for relation extraction IJCNLP-05, Jeju Island
4. Ciravegna F (2001) Adaptive information extraction from text by rule induction and generalization. In: Proceedings of the 17th IJCAI, Seattle
5. Cowie J, Lehnert W (1996) Information extraction. Commun Assoc Comput Mach 39(1):80–91
6. Downey D, Etzioni O et al (2004) Learning text patterns for web information extraction and assessment (extended version). Technical Report UW-CSE-04-05-01
7. Etzioni O, Cafarella M et al (2005) Unsupervised named-entity extraction from the Web: an experimental study. Artif Intell 165(1):91–134
8. Feldman R, Rozenfeld B et al (2006) TEG—a hybrid approach to information extraction. Knowl Inf Syst 9(1):1–18
9. Freitag D (1998) Machine learning for information extraction in informal domains. Computer Science Department, Carnegie Mellon University, Pittsburgh p 188
10. Freitag D, McCallum AK (1999) Information extraction with HMMs and shrinkage. In: Proceedings of the AAAI-99 workshop on machine learning for information extraction
11. Genkin A, Lewis DD et al (2004) Large-scale bayesian logistic regression for text categorization. DIMACS, New Brunswick pp 1–41
12. Grishman R (1996) The role of syntax in information extraction. In: Advances in Text Processing: Tipster Program Phase II. Morgan Kaufmann
13. Grishman R (1997) Information extraction: techniques and challenges. SCIE: 10–27
14. Hasegawa T, Sekine S et al (2004) Discovering relations among named entities from large corpora. ACL 2004
15. Kushmerick N, Weld DS et al (1997) Wrapper induction for information extraction. IJCAI 97:729–737
16. Li Z, Ng WK et al (2005) Web data extraction based on structural similarity. Knowl Inf Syst 8(4):438–461
17. Miller G (1990) WordNet: an on-line lexical database. Int J Lexicogr 3(4):235–312
18. Ravichandran D, Hovy E (2002) Learning surface text patterns for a question answering system. 40th ACL Conference
19. Riloff E (1993) Automatically constructing a dictionary for information extraction tasks. AAAI-93
20. Riloff E, Jones R (1999) Learning dictionaries for information extraction by multi-level boot-strapping. AAAI-99
21. Soderland S (1999) Learning information extraction rules for semi-structured and free text. Mach Learn 34(1–3):233–272
22. Wong T-L, Lam W (2007) Learning to extract and summarize hot item features from multiple auction web sites. Knowl Inf Syst

Author Biographies

Ronen Feldman is an Associate Professor of Information Systems at the Business School of the Hebrew University in Jerusalem. He received his B.Sc. in Math, Physics and Computer Science from the Hebrew University and his Ph.D. in Computer Science from Cornell University in New York. He was an Adjunct Professor at NYU Stern Business School. He is the founder of ClearForest Corporation, a Boston based company specializing in development of text mining tools and applications. He has given more than 30 tutorials on text mining and information extraction and authored numerous papers on these topics. He is the author of the book "The Text Mining Handbook" published by Cambridge University Press in 2007.

Benjamin Rozenfeld is a research scientist at Topodia Corporation. He received his B.Sc. in Mathematics and Computer Science from Bar-Ilan University. He is the co-inventor of the DIAL information extraction language.