Expanding the Recall of Relation Extraction by Bootstrapping

Junji Tomita
NTT Cyber Solutions Laboratories,
NTT Corporation
1-1 Hikarinooka Yokosuka-Shi,
Kanagawa 239-0847, Japan
tomita.junji@lab.ntt.co.jp

Stephen Soderland **Oren Etzioni**
Department of Computer Science
& Engineering
University of Washington
Seattle, WA 98195-2350
{soderlan,etzioni}
@cs.washington.edu

Abstract

Most works on relation extraction assume considerable human effort for making an annotated corpus or for knowledge engineering. Generic patterns employed in KnowItAll achieve unsupervised, high-precision extraction, but often result in low recall. This paper compares two bootstrapping methods to expand recall that start with automatically extracted seeds by KnowItAll. The first method is string pattern learning, which learns string contexts adjacent to a seed tuple. The second method learns less restrictive patterns that include bags of words and relation-specific named entity tags. Both methods improve the recall of the generic pattern method. In particular, the less restrictive pattern learning method can achieve a 250% increase in recall at 0.87 precision, compared to the generic pattern method.

1 Introduction

Relation extraction is a task to extract tuples of entities that satisfy a given relation from textual documents. Examples of relations include CeoOf(Company, Ceo) and Acquisition(Organization, Organization). There has been much work on relation extraction; most of it employs knowledge engineering or supervised machine learning approaches (Feldman et al., 2002; Zhao and Grishman, 2005). Both approaches are labor intensive.

We begin with a baseline information extraction system, KnowItAll (Etzioni et al., 2005), that does unsupervised information extraction at Web scale. KnowItAll uses a set of generic extraction patterns, and automatically instantiates rules by combining these patterns with user supplied relation labels. For example, KnowItAll has patterns for a generic "of" relation:

NP1 's <relation> , NP2

NP2 , <relation> of NP1

where NP1 and NP2 are simple noun phrases that extract values of argument1 and argument2 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.

If a user supplies the relation labels "ceo" and "chief executive officer" for the relation CeoOf(Company, Ceo), KnowItAll inserts these labels into the generic patterns shown above, to create 4 extraction rules:

NP1 's ceo , NP2

NP1 's chief executive officer , NP2

NP2 , ceo of NP1

NP2 , chief executive officer of NP1

The same generic patterns with different labels can also produce extraction rules for a MayorOf relation or an InventorOf relation. These rules have alternating context strings (exact string match) and extraction slots (typically an NP or head of an NP). This can produce rules with high precision, but low recall, due to the wide variety of contexts describing a relation. This paper looks at ways to enhance recall over this baseline system while maintaining high precision.

To enhance recall, we employ bootstrapping techniques which start with seed tuples, i.e. the most frequently extracted tuples by the baseline system. The first method represents rules with three context strings of tokens immediately adjacent to the extracted arguments: a left context,

middle context, and right context. These are induced from context strings found adjacent to seed tuples.

The second method uses a less restrictive pattern representation such as bag of words, similar to that of SnowBall(Agichtein, 2005). SnowBall is a semi-supervised relation extraction system. The input of Snowball is a few hand labeled correct seed tuples for a relation (e.g. <Microsoft, Steve Ballmer> for CeoOf relation). SnowBall clusters the bag of words representations generated from the context strings adjacent to each seed tuple, and generates rules from them. It calculates the confidence of candidate tuples and the rules iteratively by using an EM-algorithm. Because it can extract any tuple whose entities co-occur within a window, the recall can be higher than the string pattern learning method. The main disadvantage of SnowBall or a method which employs less restrictive patterns is that it requires Named Entity Recognizer (NER).

We introduce Relation-dependent NER (Relation NER), which trains an off-the-shelf supervised NER based on CRF(Lafferty et al., 2001) with bootstrapping. This learns relation-specific NE tags, and we present a method to use these tags for relation extraction.

This paper compares the following two bootstrapping strategies.

SPL: a simple string pattern learning method. It learns string patterns adjacent to a seed tuple.

LRPL: a less restrictive pattern learning method. It learns a variety of bag of words patterns, after training a Relation NER.

Both methods are completely self-supervised extensions to the unsupervised KnowItAll. A user supplies KnowItAll with one or more relation labels to be applied to one or more generic extraction patterns. No further tagging or manual selection of seeds is required. Each of the bootstrapping methods uses seeds that are automatically selected from the output of the baseline KnowItAll system.

The results show that both bootstrapping methods improve the recall of the baseline system. The two methods have comparable results, with LRPL outperforms SPL for some relations and SPL outperforms LRPL for other relations.

The rest of the paper is organized as follows. Section 2 and 3 describe SPL and LRPL respectively. Section 4 reports on our experiments, and section 5 and 6 describe related works and conclusions.

2 String Pattern Learning (SPL)

Both SPL and LRPL start with seed tuples that were extracted by the baseline KnowItAll system, with extraction frequency at or above a threshold (set to 2 in these experiments). In these experiments, we downloaded a set of sentences from the Web that contained an occurrence of at least one relation label and used this as our reservoir of unlabeled training and test sentences. We created a set of positive training sentences from those sentences that contained both argument values of a seed tuple.

SPL employs a method similar to that of (Downey et al., 2004). It generates candidate extraction rules with a prefix context, a middle context, and a right context. The prefix is zero to L_{side} tokens immediately to the left of extracted argument1, the middle context is all tokens between argument1 and argument2, and the right context of zero to L_{side} tokens immediately to the right of argument2. It discards patterns with more than L_{mid} intervening tokens or without a relation label.

SPL tabulates the occurrence of such patterns in the set of positive training sentences (all sentences from the reservoir that contain both argument values from a seed tuple in either order), and also tabulates their occurrence in negative training sentences. The negative training are sentences that have one argument value from a seed tuple and a nearest simple NP in place of the other argument value. This idea is based on that of (Ravichandran and Hovy, 2002) for a QA system. SPL learns a possibly large set of strict extraction rules that have alternating context strings and extraction slots, with no gaps or wildcards in the rules.

SPL selects the best patterns as follows:

1. Groups the context strings that have the exact same middle string.

2. Selects the best pattern having the largest pattern score, PS, for each group of context strings having the same middle string.

$$PS(p) = \frac{|S_{positive}(p)| + 1}{|S_{positive}(p)| + |S_{negative}(p)| + \alpha} \quad (1)$$

3. Selects the patterns having PS greater than $\tau_{pattern}$.

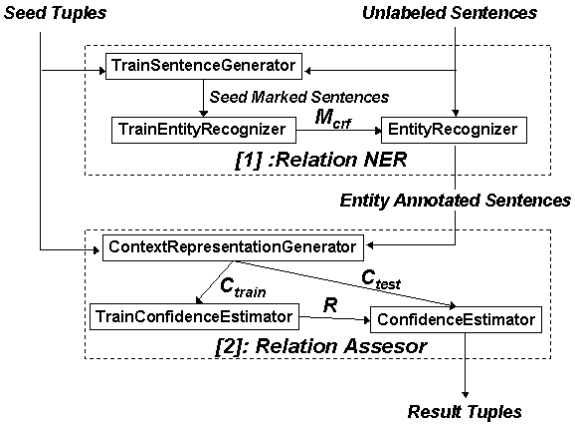

Figure 1: The architecture of LRPL (Less Restrictive Pattern Learning).

where $S_{positive}(p)$ is a set of sentences that match pattern p and include both argument values of a seed tuple. $S_{negative}(p)$ is a set of sentences that match p and include just one argument value of a seed tuple (e.g. just a company or a person for CeoOf). α is a constant for smoothing.

3 Less Restrictive Pattern Learning (LRPL)

LRPL uses a more flexible rule representation than SPL. As before, the rules are based on a window of tokens to the left of the first argument, a window of middle tokens, and a window of tokens to the right of the second argument. Rather than using exact string match on a simple sequence of tokens, LRPL uses a combination of bag of words and immediately adjacent token. The left context is based on a window of L_{side} tokens immediately to the left of argument1. It has two sets of tokens: the token immediately to the left and a bag of words for the remaining tokens. Each of these sets may have zero or more tokens. The middle and right contexts are similarly defined. We call this representation extended bag of words.

Here is an example of how LRPL represents the context of a training sentence with window size set to 4. "Yesterday , <Arg2>Steve Ballmer</Arg2>, the Chief Executive Officer of <Arg1>Microsoft</Arg1> said that he is ...".

```
order:  arg2_arg1
values: Steve Ballmer, Microsoft
L: {yesterday} {,}
M: {,} {chief executive officer the} {of}
R: {said} {he is that}
```

Some of the tokens in these bags of words may be dropped in merging this with patterns from other training sentences. Each rule also has a confidence score, learned from EM-estimation.

We experimented with simply using three bags of words as in SnowBall, but found that precision was increased when we distinguished the tokens immediately adjacent to argument values from the other tokens in the left, middle, and right bag of words.

Less restrictive patterns require a Named Entity Recognizer (NER), because the patterns can not extract candidate entities by themselves[1]. LRPL trains a supervised NER in bootstrapping for extracting candidate entities.

Figure 1 overviews LRPL. It consists of two bootstrapping modules: Relation NER and Relation Assessor. LRPL trains the Relational NER from seed tuples provided by the baseline KnowItAll system and unlabeled sentences in the reservoir. Then it does NE tagging on the sentences to learn the less restrictive rules and to extract candidate tuples. The learning and extraction steps at Relation Assessor are similar to that of SnowBall; it generates a set of rules and uses EM-estimation to compute a confidence in each rule. When these rules are applied, the system computes a probability for each tuple based on the rule confidence, the degree of match between a sentence and the rule, and the extraction frequency.

3.1 Relation dependent Named Entity Recognizer

Relation NER leverages an off-the-shelf supervised NER, based on Conditional Random Fields (CRF). In Figure 1, TrainSentenceGenerator automatically generates training sentences from seeds and unlabeled sentences in the reservoir. TrainEntityRecognizer trains a CRF on the training sentences and then EntityRecognizer applies the trained CRF to all the unlabeled sentences, creating entity annotated sentences.

It can extract entities whose type matches an argument type of a particular relation. The type is not explicitly specified by a user, but is automatically determined according to the seed tuples. For example, it can extract 'City' and 'Mayor' type entities for MayorOf(City, Mayor) relation. We describe CRF in brief, and then how to train it in bootstrapping.

[1] Although using all noun phrases in a sentence may be possible, it apparently results in low precision.

3.1.1 Supervised Named Entity Recognizer

Several state-of-the-art supervised NERs are based on a feature-rich probabilistic conditional classifier such as Conditional Random Fields (CRF) for sequential learning tasks(Lafferty et al., 2001; Rosenfeld et al., 2005). The input of CRF is a feature sequence X of features x_j, and outputs a tag sequence T of tags t_j. In the training phrase, a set of $< X_i, T_i >$ is provided, and outputs a model M_{crf}. In the applying phase, given X, it outputs a tag sequence T by using M_{crf}. In the case of NE tagging, given a sequence of tokens, it automatically generates a sequence of feature sets; each set is corresponding to a token. It can incorporate any properties that can be represented as a binary feature into the model, such as words, capitalized patterns, part-of-speech tags and the existence of the word in a dictionary. It works quite well on NE tagging tasks (McCallum and Li, 2003).

3.1.2 How to Train Supervised NER in Bootstrapping

We use bootstrapping to train CRF for relation-specific NE tagging as follows: 1) select the sentences that include all the entity values of a seed tuple, 2) automatically mark the argument values in each sentence, and 3)train CRF on the seed marked sentences. An example of a seed marked sentence is the following:

```
seed tuple:  <Microsoft, Steve Ballmer>

seed marked sentence:
"Yesterday, <Arg2>Steve Ballmer</Arg2>,
CEO of <Arg1>Microsoft</Arg1>
announced that ..."
```

Because of redundancy, we can expect to generate a fairly large number of seed marked sentences by using a few highly frequent seed tuples. To avoid overfitting on terms from these seed tuples, we substitute the actual argument values with random characters for each training sentence, preserving capitalization patterns and number of characters in each token.

3.2 Relation Assessor

Relation Assessor employs several SnowBall-like techniques including making rules by clustering and EM-estimation for the confidence of the rules and tuples.

In Figure 1, ContextRepresentationGenerator generates extended bag of words contexts, from entity annotated sentences, and classifies the contexts into two classes: training contexts C_{train} (if their entity values and their orders match a seed tuple) and test contexts C_{test} (otherwise). TrainConfidenceEstimator clusters C_{train} based on the match score between contexts, and generates a rule from each cluster, that has average vectors over contexts belonging to the cluster. Given a set of generated rules R and test contexts C_{test}, ConfidenceEstimator estimates each tuple confidence in C_{test} by using an EM algorithm. It also estimates the confidence of the tuples extracted by the baseline system, and outputs the merged result tuples with confidence.

We describe the match score calculation method, the EM-algorithm, and the merging method in the following sub sections.

3.2.1 Match Score Calculation

The match score (or similarity) M of two extended bag of words contexts c_i, c_j is calculated as the linear combination of the cosine values between the corresponding vectors.

$$M(c_i, c_j) = \sum_{s,t} w_{st} cos(c_{ist}, c_{jst}) \qquad (2)$$

where, s is the index of left, middle, or right contexts. t is the index of left adjacent, right adjacent, or other tokens. w_{st} is the weight corresponding to the context vector indexed by s and t.

To achieve high precision, Relation Assessor uses only the entity annotated sentences that have just one entity for each argument (two entities in total) and where those entities co-occur within L_{mid} tokens window, and it uses at most L_{side} left and right tokens. It discards patterns without a relation label.

3.2.2 EM-estimation for tuple and rule confidence

Several rules generated from only positive evidence result in low precision (e.g. rule "of" for MayorOf relation generated from "Rudolph Giuliani of New York"). This problem can be improved by estimating the rule confidence by the following EM-algorithm.

1. For each c_{ij} in C_{test}, identifies the best match rule $r^*(c_{ij})$, based on the match score between c_{ij} and each rule r. c_{ij} is the jth context that includes tuple t_i.

$$r^*(c_{ij}) = \text{argmax}_r M(r, c_{ij}) \qquad (3)$$

2. Initializes seed tuple confidence, $TC(t_s) = 1$ for all t_s, where t_s is a seed tuple.

3. Calculates tuple confidence, TC, and rule confidence, RC, by using EM-algorithm. E and M stages are iterated several times.

E stage:

$$RC(r_k) = \frac{\sum_u TC(t_u(r_k)) + 1}{numOfMatch(r_k) + \beta} \quad (4)$$

M stage:

$$TC(t_i) = \quad (5)$$
$$1 - \prod_j (1 - RC(r^*(c_{ij}))M(r^*(c_{ij}), c_{ij})) \quad (6)$$

where

$$TC(t_u(r_k)) = \begin{cases} TC(t_u) & if \quad r_k = r^*(c_{uj}) \quad \exists j \\ 0 & otherwise \end{cases}$$

β is a constant for smoothing.

This algorithm assigns a high confidence to the rules that frequently co-occur with only high confident tuples. It also assigns a high confidence to the tuples that frequently co-occur with the contexts that match high confidence rules.

When it merges the tuples extracted by the baseline system, the algorithm uses the following constant value for any context that matches a baseline pattern.

$$RC(r_b)M(r_b, c_{ib}) = \max_c RC(r^*(c))M(r^*(c), c) \quad (7)$$

where c_{ib} denotes the context of tuple t_i that matches a baseline pattern, and r_b is any baseline pattern. With this calculation, the confidence of any tuple extracted by a baseline pattern is always greater than or equal to that of any tuple that is extracted by the learned rules and has the same frequency.

4 Evaluation

The focus of this paper is the comparison between bootstrapping strategies for extraction, i.e., string pattern learning and less restrictive pattern learning having Relation NER. Therefore, we first compare these two bootstrapping methods with the baseline system. Furthermore, we also compare Relation NER with a generic NER, which is trained on a pre-existing hand annotated corpus.

Table 1: Weights corresponding to a context vector (w_{st}).

		adjacency			
		left	other	right	total
context	left		0.067	0.133	0.2
	middle	0.24	0.12	0.24	0.6
	right	0.133	0.067		0.2

4.1 Relation Extraction Task

We compare SPL and LRPL with the baseline system on 5 relations: Acquisition, Merger, CeoOf, MayorOf, and InventorOf. We downloaded about from 100,000 to 220,000 sentences for each of these relations from the Web, which contained a relation label (e.g. "acquisition", "acquired", "acquiring" or "merger", "merged", "merging"). We used all the tuples that co-occur with baseline patterns at least twice as seeds. The numbers of seeds are between 33 (Acquisition) and 289 (CeoOf).

For consistency, SPL employs the same assessment methods with LRPL. It uses the EM algorithm in Section 3.2.2 and merges the tuples extracted by the baseline system. In the EM algorithm, the match score $M(p, t)$ between a learned pattern p and a tuple t is set to a constant τ_{match}.

LRPL uses MinorThird (Cohen, 2004) implementation of CRF for Relation NER. The features used in the experiments are the lower-case word, capitalize pattern, part of speech tag of the current and +-2 tokens, and the previous state (tag) referring to (Minkov et al., 2005; Rosenfeld et al., 2005). The parameters used for SPL and LRPL are experimentally set as follows: $\tau_{pattern} = 0.5$, $\tau_{match} = 0.6$, $L_{mid} = 5$, $L_{side} = 3$, $\alpha = 3$, $\beta = 3$ and the context weights for LRPL shown in Table 1.

Figure 2-6 show the recall-precision curves. We use the number of correct extractions to serve as a surrogate for recall, since computing actual recall would require extensive manual inspection of the large data sets. Compared to the the baseline system, both bootstrapping methods increases the number of correct extractions for almost all the relations at around 80% precision. For MayorOf relation, LRPL achieves 250% increase in recall at 0.87 precision, while SPL's precision is less than the baseline system. This is because SPL can not distinguish correct tuples from the error tuples that

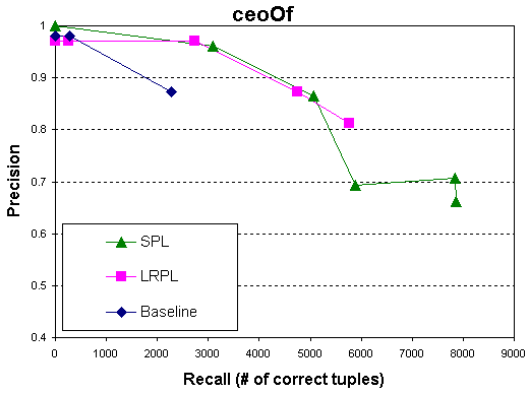

Figure 2: The recall-precision curve of CeoOf relation.

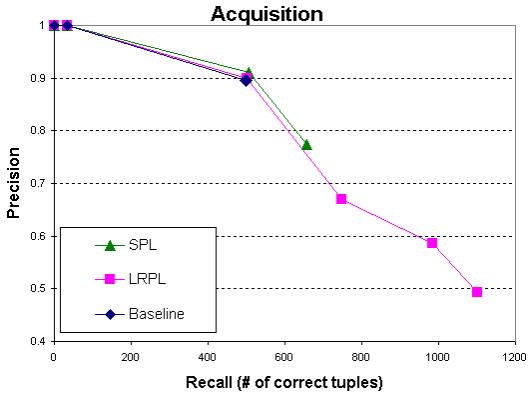

Figure 4: The recall-precision curve of Acquisition relation.

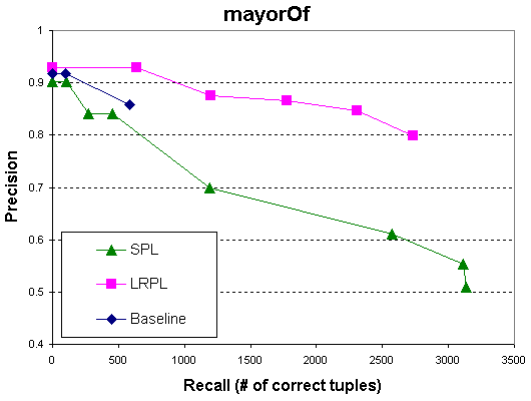

Figure 3: The recall-precision curve of MayorOf relation.

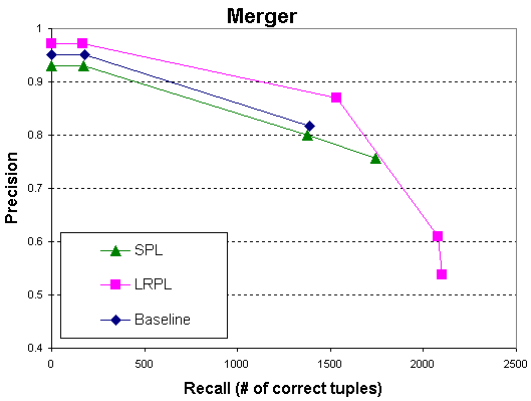

Figure 5: The recall-precision curve of Merger relation.

co-occur with a short strict pattern, and that have a wrong entity type value. An example of the error tuples extracted by SPL is the following:

```
Learned Pattern: NP1 Mayor NP2
Sentence:
 "When Lord Mayor Clover Moore spoke,..."
Tuple:  <Lord, Clover Moore>
```

The improvement of Acquisition and Merger relations is small for both methods; the rules learned for Merger and Acquisition made erroneous extractions of mergers of geo-political entities, acquisition of data, ball players, languages or diseases. For InventorOf relation, LRPL does not work well. This is because 'Invention' is not a proper noun phrase, but a noun phrase. A noun phrase includes not only nouns, but a particle, a determiner, and adjectives in addition to non-capitalized nouns. Our Relation NER was unable to detect regularities in the capitalization pattern and word length of invention phrases.

At around 60% precision, SPL achieves higher recall for CeoOf and MayorOf relations, in contrast, LRPL achieves higher recall for Acquisition and Merger. The reason can be that nominal style relations (CeoOf and MayorOf) have a smaller syntactic variety for describing them. Therefore, learned string patterns are enough generic to extract many candidate tuples.

4.2 Entity Recognition Task

Generic types such as person, organization, and location cover many useful relations. One might expect that NER trained for these generic types, can be used for different relations without modifications, instead of creating a Relation NER. To show the effectiveness of Relation NER, we compare Relation NER with a generic NER trained on a pre-existent hand annotated corpus for generic types; we used MUC7 train, dry-run test, and formal-test documents(Table 2) (Chinchor, 1997). We also incorporate the following additional knowledge into the CRF's features referring to (Minkov et al., 2005; Rosenfeld et al.,

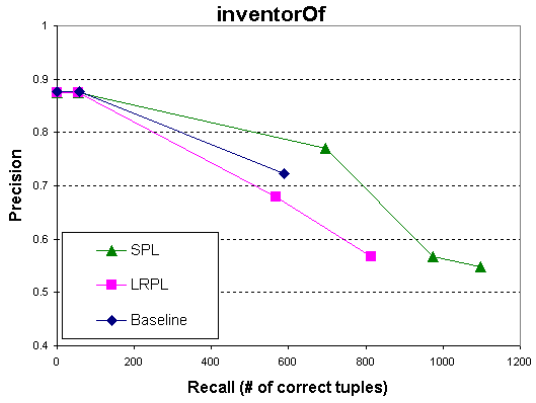

Figure 6: The recall-precision curve of InventorOf relation.

Table 2: The number of entities and unique entities in MUC7 corpus. The number of documents is 225.

entity	all	uniq
Organization	3704	993
Person	2120	1088
Location	2912	692

2005): first and last names, city names, corp designators, company words (such as "technology"), and small size lists of person title (such as "mr.") and capitalized common words (such as "Monday"). The base features for both methods are the same as the ones described in Section 4.1.

The ideal entity recognizer for relation extraction is recognizing only entities that have an argument type for a particular relation. Therefore, a generic test set such as MUC7 Named Entity Recognition Task can not be used for our evaluation. We randomly selected 200 test sentences from our dataset that had a pair of correct entities for CeoOf or MayorOf relations, and were not used as training for the Relation NER. We measured the accuracy as follows.

$$Recall_{arg} = \frac{|E_{extracted} \cap E_{true,arg}|}{|E_{true,arg}|} \quad (8)$$

$$Precision_{arg} = \frac{|E_{extracted} \cap E_{true,arg}|}{|E_{extracted}|} \quad (9)$$

where, $E_{true,arg}$ is a set of true entities that have an argument type of a target relation. $E_{extracted}$ is a set of entities extracted as an argument.

Because Relation NER is trained for argument types (such as 'Mayor'), and the generic NER is trained for generic types (such as person), this calculation is in favor of Relation NER. For fair comparison, we also use the following measure.

$$Precision_{generic} = \frac{|E_{extracted} \cap E_{true,generic}|}{|E_{extracted}|} \quad (10)$$

where, $E_{true,generic}$ is a set of true entities that have a generic type [2].

Table 3 shows that the Relation NER consistently works better than the generic NER, even when additional knowledge much improved the recall. This suggests that training a Relation NER for each particular relation in bootstrapping is better approach than using a NER trained for generic types.

5 Related Work

SPL is a similar approach to DIPRE (Brin, 1998) DIPRE uses a pre-defined simple regular expression to identify argument values. Therefore, it can also suffer from the type error problem described above. LRPL avoids this problem by using the Relation NER.

LRPL is similar to SnowBall(Agichtein, 2005), which employs a generic NER, and reported that most errors come from NER errors. Because our evaluation showed that Relation NER works better than generic NER, a combination of Relation NER and SnowBall can make a better result in other settings. [3]

(Collins and Singer, 1999) and (Jones, 2005) describe self-training and co-training methods for Named Entity Classification. However, the problem of NEC task, where the boundary of entities are given by NP chunker or parser, is different from NE tagging task. Because the boundary of an entity is often different from a NP boundary, the technique can not be used for our purpose; "Microsoft CEO Steve Ballmer" is tagged as a single noun phrase.

6 Conclusion

This paper describes two bootstrapping strategies, SPL, which learns simple string patterns, and LRPL, which trains Relation NER and uses it with less restrictive patterns. Evaluations showed both

[2] Although $Recall_{generic}$ can be defined in the same way, we did not use it, because of our purpose and much effort needed for complete annotation for generic types.

[3] Of course, further study needed for investigating whether Relation NER works with a smaller number of seeds.

Table 3: The argument precision and recall is the average over all arguments for CeoOf, and MayorOf relations. The Location is for MayorOf, Organization is for CeoOf, and person is the average of both relations.

	Argument			Location	Organization	Person
	Recall	Precision	F	Precision	Precision	Precision
R-NER	0.650	0.912	0.758	0.922	0.906	0.955
G-NER	0.392	0.663	0.492	0.682	0.790	0.809
G-NER+dic	0.577	0.643	0.606	0.676	0.705	0.842

methods enhance the recall of the baseline system for almost all the relations. For some relations, SPL and LRPL have comparable recall and precision. For InventorOf, where the invention is not a named entity, SPL performed better, because its patterns are based on noun phrases rather than named entities.

LRPL works better than SPL for MayorOf relation by avoiding several errors caused by the tuples that co-occur with a short strict context, but have a wrong type entity value. Evaluations also showed that Relation NER works better than the generic NER trained on MUC7 corpus with additional dictionaries.

Acknowledgements

This work was done while the first author was a visiting Scholar at the University of Washington. The work was carried out at the University's Turing Center and was supported in part by NSF grant IIS-0312988, DARPA contract NBCHD030010, ONR grant N00014-02-1-0324, and a gift from Google. We would like to thank Dr. Eugene Agichtein for informing us the technical details of SnowBall, and Prof. Ronen Feldman for a helpful discussion.

References

Eugene Agichtein. 2005. *Extracting Relations From Large Text Collections*. Ph.D. thesis, Columbia University.

Sergey Brin. 1998. Extracting Patterns and Relations from the World Wide Web. In *WebDB Workshop at EDBT'98*, pages 172–183, Valencia, Spain.

Nancy Chinchor. 1997. Muc-7 named entity task definition version 3.5.

William W. Cohen. 2004. Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data.

Michael Collins and Yoram Singer. 1999. Unsupervised models for named entity classification. In *EMNLP 99*.

Doug Downey, Oren Etzioni, Stephen Soderland, and Daniel S. Weld. 2004. Learning text patterns for web information extraction and assessment. In *AAAI 2004 Workshop on ATEM*.

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2005. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134.

Ronen Feldman, Yonatan Aumann, Michal Finkelstein-Landau, Eyal Hurvitz, Yizhar Regev, and Ariel Yaroshevich. 2002. A comparative study of information extraction strategies. In *CICLing*, pages 349–359.

Rosie Jones. 2005. *Learning to Extract Entities from Labeled and Unlabeled Texts*. Ph.D. thesis, CMU-LTI-05-191.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01*, pages 282–289.

Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *CoNLL-2003)*.

Einat Minkov, Richard C. Wang, and William W. Cohen. 2005. Extracting personal names from email: Applying named entity recognition to informal text. In *EMNLP/HLT-2005*.

D. Ravichandran and D. Hovy. 2002. Learning Surface Text Patterns for a Question Answering System. In *Procs. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 41–47, Philadelphia, Pennsylvania.

Binyamin Rosenfeld, Moshe Fresko, and Ronen Feldman. 2005. A systematic comparison of feature-rich probabilistic classifiers for ner tasks. In *PKDD*, pages 217–227.

Shubin Zhao and Ralph Grishman. 2005. Extracting relations with integrated information using kernel methods. In *ACL'05*, pages 419–426, June.