A Survey and Exploration of Relation Extraction in Active Learning Systems

Alen Lukic
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alukic@cs.cmu.edu

Abstract

The reliable recognition and extraction of semantic relations between entities is important to many applications in the domain of natural language processing and information retrieval, such as search engines and question-answering systems. In this survey I summarize the primary relation extraction approaches presented in the literature, particularly focusing on weakly supervised bootstrap methods. I offer a critique and discuss possible improvements of the different methodologies used at various points in the relation extraction process. Finally, I discuss an extension of relation extraction to an active, unsupervised ontology-learning system in the context of a configuration space exploration framework.

1 Introduction

A vast amount of unstructured and semi-structured text information exists on the Web in a growing number of forms, including electronic books, academic journals, news articles, blogs and social media feeds, and e-mail communications. In order to perform any sort of computational analysis on such corpora, relevant documents must be retrieved, and information of interest must be reliably recognized, extracted, annotated and stored. Many methods exist for generating different types of annotations over sections of unstructured text, such as part-of-speech tagging, named entity recognition, relation extraction, and shallow parsing. This survey concentrates on relation extraction: the process of recognizing relationships between sets of entities in unstructured or semi-structured text.

A relation is defined as a tuple $t = (e_1, ..., e_n)$, where the tuple elements are the entities which participate in the relation. The following is an example of a binary relation which expresses that Subra Suresh is the president of Carnegie Mellon University: *president-of(Subra Suresh, CMU)* - another common notation is *(Subra Suresh, president-of, CMU)*. Higher-order relations are also possible. For example, a ternary relation which expresses that Subra Suresh became the president of Carnegie Mellon in February 2013 is *became-president(Subra Suresh, CMU, February 2013)*. Most commonly, relation extraction approaches described in the literature focus on binary relations.

The approaches of interest in this survey are bootstrap seeding methods, semi-supervised relation classification, and active feedback loops designed for an iteratively improving extraction process. In Section 2, I will briefly discuss the history of relation extraction up to the emergence of these techniques.

In Section 3, I will compare and contrast approaches which fundamentally implement Yarowsky's algorithm, which I will summarize in Section 2 [**?**]. I will discuss any pre-processing which the system performs on either the seeds or the corpus of interest, compare the instance extraction methodologies of the systems, analyze the systems' fitness metrics for learned relation patterns (where applicable) and extracted relation instances, assess whether the system utilizes a feedback loop at any point in the extraction

process and, if applicable, how this affects system performance.

The purpose of this survey is to lay the groundwork for the incorporation of an appropriate and effective relation extraction approach in an active, unsupervised ontology-learning system with configurable component parameters. I will describe this system and its framework in Section 4 and discuss how to incorporate my conclusions from Section 3 into this system.

2 Previous Work

Many earlier approaches modeled relation extraction as a supervised classification problem. A binary classifier, such as Naive Bayes, Logistic Regression, Voted Perceptron or Support Vector Machines (SVM), is trained on labeled training instances and used to classify potential relation instances. Approaches vary by how the feature vectors used in classification are constructed.

One feature selection technique involves extracting syntactic and semantic features from the text surrounding the entities which participate in the relation. Examples of syntactic features are the entities themselves, the sequence of words between the entities, and the path in the parse tree which contains the entities. An example of a semantic feature is the dependency tree path between the entities. Zhao & Grishman utilize these features in an SVM model with a polynomial kernel for the purpose of relation extraction [2], while GuoDong et. al. do the same but with a linear kernel [3].

The primary limitation of using semantic and syntactic features is that feature selection is heuristic and often suboptimal. The use of some features degrades the performance of the classifier, and selecting an optimal set of features is difficult. An approach which resolves this difficulty involves the use of string kernels in SVM classifiers which implicitly explore the full richness of the input representation of the text in a higher-dimensional space. The basis of this approach is the substring kernel described in the paper by Lodhi et. al. [4]. This kernel computes the similarity between two strings by determining the number of common subsequences they share. However, the computation may be extended to the general case of computing the structural similarity between any two objects, such as word sequences and parse trees. Bunescu & Mooney use this kernel by computing the similarity between the word sequences before, between, and following the two entities; the kernel is simply the sum of these similarities. Using an SVM classifier, they achieve both superior precision and recall over an existing rule-based method on a data set of MEDLINE abstracts [5].

Zelenko et. al. apply the aforementioned kernel to shallow parse subtrees derived from a shallow parse of the sentence containing the two potentially related entities. The similarity metric is computed by calculating the total number attributes shared by subtrees which cover both of the entities [6]. Culotta & Sorensen use a similar approach but utilize dependency parse trees instead of shallow parse trees, arguing that the richer structure augments performance [7]. These kernel computations are computationally expensive at $O(mn^3)$ complexity, given that m and n are the number of nodes in the parse trees which are being compared. Bunescu & Mooney argue that comparing the shortest paths between the entities in the dependency tree is sufficient, as this path contains sufficient information to encode the relation between them. Not only is the computational complexity of this kernel linear in the size of the tree nodes, but in fact, this approach outperforms the shallow and dependency parse kernels in terms of recall and achieves similar precision [8].

Supervised methods are largely limited by the availability of and expense of obtaining reliable labeled training data. Semi-supervised bootstrap methods seek to overcome this problem. Bootstrapping in the context of relation extraction applies the idea behind Yarowsky's algorithm. On a high level, the algorithm operates as follows:

1. Train a classifier with an initial seed set.
2. Extract new instances and label them with the classifier.
3. Add positive instances which meet a requisite confidence threshold to the training set.
4. Repeat 1 - 3 until some defined convergence criterion is met.

In Section 3, I will analyze the various components of several relation extraction systems which effectively implement Yarowsky's algorithm. In particular, I will discuss the following works:

1. Pantel & Pennachiotti, 2006 (Espresso) [9]

2. Rozenfeld & Feldman, 2008 (Self-Supervised Relation Extraction System) [10]

3. Carlson, Betteridge, Wang, Hruschka Jr., & Mitchell, 2010 (Meta Bootstrap Learner) [11]

Note that for the systems which do not utilize an active feedback loop, I consider these systems to implement a special case of Yarowsky's algorithm whose convergence criterion is a single repetition of steps 1 - 3.

3 Semi-Supervised Bootstrap Approaches

3.1 Pre-Processing

3.1.1 Espresso

Espresso utilizes the pattern-learning algorithm described in the work by Ravichandran and Hovy in order to induce a set of generic patterns for instance extraction from the seed relations [12]. The patterns are ranked by a reliability metric and only the top k patterns are selected, where k is one greater than the number of patterns from the previous iteration. New statistical evidence may result in patterns being discarded.

3.1.2 Self-Supervised Relation Extraction System (SRES)

SRES utilizes a set of target relation definitions as its input. The Sentence Gatherer system component downloads a large number of web documents which may contain relations of interest using a WordNet-augmented version of its target relation set (the authors do not specify how; I will assume a search engine is used). Then, for each target relation, the Seed Gatherer sorts each sentence in these documents into a positive and negative set by checking the entities and context words in each sentence against the relation definition. If the relation is defined to be symmetric, it switches the order of the entities in the sentence and places that variant of the sentence into the relation's positive set as well; otherwise, if the relation is antisymmetric, it places that sentence variant into the negative set.

Next, the Pattern Learner generates a set of generic patterns by using a dynamic programming optimal string alignment algorithm for each pair of sentences in each relation's positive set. This produces a very large amount of patterns, which are filtered according to a relevance heuristic (discussed later).

3.1.3 Meta Bootstrap Learner (MBL)

MBL is a meta-algorithm which uses the results of two component algorithms, Coupled Pattern Learner and Coupled SEAL (discussed later). The input to both algorithms is a seed set of relation patterns and instances. Neither of these algorithms perform any sort of pre-processing on the corpus or the seed set.

3.1.4 Discussion

The extent of the pre-processing which a relation extraction system performs plays a large role in its time complexity and accuracy. MBL clearly outperforms the other 2 systems in this phase; the system avoids expending any time on pre-processing altogether and also avoids introducing error to the system.

Assume that r is both the number of seed relations given as input to Espresso and the number of target relations given as input to SRES. Assume that s is the total number of sentences retrieved by a search engine for both systems in pre-processing, and that n is the average sentence length. Based on the description of the pattern-learning algorithm which Espresso employs and using the assumption that Ukkonen's suffix tree construction algorithm is used, the time complexity of Espresso's preprocessing phase is:

$$O(rns)$$

Assume p is the number of unfiltered patterns generated by SRES's Pattern Learner. Using the assumption that the Smith-Waterman string alignment algorithm is used, the time complexity of SRES's pre-processing phase is:

$$O(rs + n^2 s^2)$$

Assume that $s \gg r$. The above complexities can then be reduced to $O(ns)$ and $O(n^2 s^2 + p)$, respectively. This confirms the intuition that SRES's pre-processing stage is more expensive than Espresso's.

SRES also has a higher risk of introducing error in this phase of the relation extraction process than Espresso does. Although both systems use scoring metrics (described later) for their induced patterns, Espresso's pattern set can change on a later iteration of the system based on new statistical evidence, thus potentially removing any incorrect induced patterns. SRES does not utilize a feedback system, and hence any erroneous patterns induced during pre-processing will permanently affect the end system performance.

3.2 Instance Extraction

3.2.1 Espresso

Espresso's instance extraction algorithm is straightforward. The system walks the corpus and extracts instances I which match any of the patterns in P. These instances are filtered by calculating a reliability score for each of them and then selecting the top k instances (discussed later).

3.2.2 SRES

SRES utilizes the shallow parser from the OpenNLP package in order to identify noun phrases in sentences in the corpus and attempts to match them to each of the learned patterns.

3.2.3 MBL

The first phase of the Coupled Pattern Learner (CPL), an algorithm built specifically for MBL, extracts new relations based on recently retained patterns and instances (in the first iteration, it uses the seed patterns/instances) using a part-of-speech tagger. Relation instances are extracted if they match a recently promoted pattern; relation patterns are extracted if both arguments of a recently promoted instance are found in a sentence and the word sequence between them contains five or fewer tokens.

MBL also utilizes a modified version of SEAL [13], an algorithm for extracting wrappers specific to semi-structured documents such as HTML pages, called Coupled SEAL (CSEAL). The first phase of Coupled SEAL passes recently promoted relation instances as seeds to the existing SEAL algorithm in order to learn wrappers specific to each HTML page. These learned wrappers are then used in order to extract new relation instances from the HTML page.

The results of both CPL and CEAL are combined and filtered (discussed later).

3.2.4 Discussion

Notably, all three systems utilize shallow matching techniques for extracting both relation patterns and relation instances.

The authors do not specify which pattern-matching algorithm Espresso utilizes. Assume there are k patterns of average length m and d documents in the corpus, each containing s sentences average length n. I will assume that a pre-trained linear-time shallow parser and linear-time string-matching algorithm such as Rabin-Karp is utilized, whose average-case complexity is $O(m+n)$. The complexity of parsing a sentence is $O(n)$, as is finding the noun phrases. The complexity of pattern-matching for each pattern, again assuming the Rabin-Karp algorithm is used, is $O(m+n)$. Hence, the average-case complexity of SRES's instance extraction phase is $O(ds(2n+k(m+n)))$, or, simplified:

$$O(dsk(m+n)))$$

Since the authors explicitly state that SRES utilizes OpenNLP's pre-trained shallow parser, I assume that SRES extracts relation instances in the same manner as Espresso. Hence, the complexity of its instance extraction phase matches that of Espresso.

For MBL, the complexity of the instance extraction phase will be the summation of the complexity of its individual algorithms. I assume that CPL uses a pre-trained part-of-speech tagger and, as such, that the complexity of POS tagging is linear in the size of each sentence. Again, I assume Rabin-Karp is the string-matching algorithm used for pattern-matching in sentences. As such, the average-case complexity of CPL's instance extraction phase is equivalent to Espresso's.

CSEAL first calls the original SEAL algorithm in order to learn the wrappers for each HTML page. Assume r seeds of average length q are passed as input to SEAL, that there are d HTML pages and that each HTML page has, on average, s sentences of average length n. The complexity of SEAL is $O(dsr(q+n))$. For brevity, the

asymptotic complexity analysis of the SEAL algorithm is omitted, but may be derived from the original paper [14]. Once SEAL has learned the wrappers, CSEAL performs instance extraction on each page using the wrappers. Assuming w wrappers are extracted and a shallow approach similar to CPL is used, the complexity of the extraction is $O(dsw(q+n))$. The average-case complexity of CSEAL is then $O(ds(w+r)(q+n))$, and the average-case complexity of MBL's instance extraction phase as a whole is simply the combination of CPL's and CSEAL's complexities:

$$O(ds(k(m+n)+(w+r)(q+n))$$

On the whole, the complexity of the instance extraction methods is comparable; in practice, MBL's component algorithms could run in parallel, thereby effectively reducing MBL's instance extraction complexity to that of Espresso and SRES.

MBL utilizes information from previous phases of the extraction algorithm to extract new patterns and instances; Espresso utilizes similar information in its pre-processing phase. SRES's lack of a feedback loop quashes its ability to use new information in order to modify its extraction scheme and pattern/instance sets.

3.3 Confidence Estimation

3.3.1 Espresso

Espresso's confidence estimates for both patterns (P) and relation instances (I) depend on the discounted pointwise mutual information defined between each instance $i = \{x, y\} \in I$ and pattern $p \in P$:

$$pmi(i,p) = \log \frac{|x,p,y|}{|x,*,y||*,p,*|} d$$

The discount factor d offsets the bias of pointwise mutual information toward rare events; its definition, along with further discussion, may be found in the original paper [15].

The confidence (or, in the original terminology, reliability) of each pattern p is then defined as

$$r_\pi(p) = \sum_{i \in I} \frac{\frac{pmi(i,p)}{max_{pmi}} * r_l(i)}{|I|}$$

and the confidence of each instance i is defined as

$$r_l(i) = \sum_{p \in P} \frac{\frac{pmi(i,p)}{max_{pmi}} * r_\pi(p)}{|P|}$$

Notice that the two metrics are recursively defined. The base case is the confidence score of each instance in the initial seed set, which is defined as 1. The value max_{pmi} is defined as the maximum pointwise mutual information score between all pairs of patterns and instances.

The confidence score of each pattern and instance is recalculated on each iteration, and the size of both P and I grows by 1 on each iteration. This guarantees the addition of at least one new instance and pattern per iteration. Note that patterns and instances from previous iterations may also be discarded if their recomputed confidence scores are too low.

3.3.2 SRES

The learned patterns are pruned by removing stopwords from all patterns and then using WordNet to generate a set of relevant words for each target relation by following all links from the predicate to a depth of 2. All patterns which do not contain one of the relevant words are removed.

After pruning, each pattern p is assigned a score. Define P_s as the set of instances in the positive set which p matches, and P_n as the set of instances in the negative set which p matches. Then, the score is defined as

$$score(p) = \frac{|P_s|}{(|P_n|+1)^2}$$

All patterns with a score less than 6 are automatically discarded. At most, the top 300 remaining patterns are retained.

Instances are assessed using a Bayesian binary regression classifier. A set of feature vectors F_{e,P_e} composed from corpus statistics about the instance e and each of its extracting patterns $p \in P_e$ is constructed for each instance. Full details regarding the features which compose these vectors can be found in section 3.3 of the original paper.

The classifier is trained by extracting instances from a small subset of the corpus using a single gold-standard pattern. The extracted instances are manually assigned binary correctness labels.

Define the trained classifier as $L(f)$. The score of an extracted instance e is defined simply as the maximum value computed by applying the classification function to each feature vector $f_{e,p} \in F_{e,P_e}$; that is:

$$score(e) = argmax(L(f_{e,p}) \mid f_{e,p} \in F_{e,P_e})$$

Finally, all instances are passed through a named entity recognition (NER) filter. The NER determines the entity type for each of the arguments in the relation instance. If the entity types do not match those indicated by the extracting patterns, the instance is discarded.

3.3.3 MBL

MBL depends on mutual exclusion between patterns and type checking of instances in order to assign confidence estimates to patterns and instances. The input to the relation extraction system defines a set of patterns which are mutually exclusive with each other. Define the set of all patterns for which mutual exclusivity with a pattern p holds as E_p. Furthermore, the argument types of all candidate instances must match those of the extracting pattern. Define the set of all instances whose argument types do not match those of i as E_i.

The CPL algorithm filters both extracted patterns and instances. Call the set of recently promoted and candidate patterns P_p and P_c, respectively, and the set of recently promoted and candidate instances I_p and I_c, respectively. Define $count(i, p)$ as the frequency of co-occurrence between instance i and pattern p in the corpus (that is, the number of times p extracts i). A candidate instance $i_c \in I_c$ is retained if

$$\exists\{p_p \in P_p \mid count(i_c, p_p) \geq 3 * count(i_c, p_e)\}$$

That is, there must exist some recently promoted pattern p_p which co-occurs at least three times more frequently with i_c than do all of its mutually exclusive patterns. Candidate patterns are similarly filtered. A candidate pattern $p_c \in P_c$ is retained if

$$\exists\{i_p \in I_p \mid count(i_p, p_c) \geq 3 * count(i_e, p_c)\}$$

After filtering, the remaining candidate patterns and instances are ranked. Instances are ranked simply by the number of recently promoted patterns they co-occur with; that is

$$rank(i_c) \propto |count(i_c, p_p) > 0, \forall p_p \in P_p|$$

Patterns are ranked using an estimate of their precision, defined as

$$precision(p_c) = \frac{\sum_{i_c \in I_c} count(i_c, p_c)}{count(*, p)}$$

At most, CPL promotes 100 instances and 5 patterns based on the rankings. Patterns and instances must co-occur with at least 2 promoted instances and patterns, respectively, to be promoted, and must meet mutual exclusion and type-checking constraints.

CSEAL similarly ranks extracted instances by the number of unfiltered wrappers $w \in W$ which extracted them; that is

$$rank(i_c) \propto |count(i_c, w) > 0, \forall w \in W|$$

Like CPL, CSEAL ensures mutual exclusion and type constraints are met for each candidate before promoting it, and at most 100 candidate relations are promoted.

MBL utilizes a very simple heuristic: any relation or instance promoted by both CPL and CSEAL is promoted by MBL.

3.3.4 Discussion

The most computationally expensive part of Espresso's confidence estimation phase is the calculation of the maximum PMI value. In fact, in order to determine this value, the system must calculate the PMI between every currently retained instance and pattern pair - these values may be stored to avoid unnecessary recalculation. The complexity of this task is $O(|P||I|)$. This is also the asymptotic complexity of the entire phase, as the calculation of each confidence score is $O(|P|)$ and $O(|I|)$ for patterns and instances, respectively, and the rank calculation complexity is $O(|P| \log |P|)$ and $O(|I| \log |I|)$, and these are lower-order terms.

The complexity of SRES's confidence estimation phase is more difficult to determine. The determination of relevant words for each pattern p will depend on how many WordNet nodes for each word in the pattern are within a distance of at most 2 for that word. For simplicity, let's

assume that on average there exist $O(|p|)$ such nodes and that the average length of each pattern is m. Then the complexity of the pruning state is $O(|P||p|m)$. Next, the remaining patterns are assigned a score and ranked. Each pattern must be checked against each sentence in the target relation's positive and negative sets. If there are s total sentences of average length n, then the complexity of scoring and ranking is $O(|P|sn)$. Following scoring, a Bayesian binary regression classifier is trained on instances extracted from a subset of the corpus using a single pattern. Assuming e instances are extracted, then the complexity of training the classifier is $O(e)$. If I is the set of instances extracted by SRES in the instance extraction phase, then prediction takes $O(|I|)$ time. Assume $|I| \gg e$. Then the total complexity of SRES's confidence estimation phase is $O(|P|(|p|m + sn) + |I|)$.

3.4 Feedback System

3.4.1 Espresso

Espresso's feedback system is manifested in selection and expansion strategies. The system grows the size of its instance and pattern sets based on the current iteration. Furthermore, the confidence estimation score of each pattern and instance is recalculated on each iteration. This creates a dynamic set of instances and patterns in which an element may be dropped on any given iteration, depending on the results of the previous iteration.

The number of instances extracted on a particular information may provide too little statistical evidence to ensure valid pattern discovery in the next iteration, especially in small corpora. If this occurs, Espresso employs two instance expansion strategies: web expansion and syntactic expansion.

In web expansion, for each $i = \{x, y\} \in I$, and for each pattern $p = \{X, t, Y\} \in P$, where t is the text which connects the two entities in the relation, Espresso issues a query to Google of the form $x\ t\ *$. The results are then added to the candidate instances for the current iteration and filtered as usual.

In syntactic expansion, new instances are created by extracting sub-expressions from the text which connects the two entities in the relation and then re-connecting the entities with these subexpressions.

3.4.2 SRES

SRES runs in a single iteration and therefore does not utilize a feedback strategy.

3.4.3 MBL

The feedback system of MBL is seen in the relationship between its instance extraction phase and its promotion phase. Relation instances which match a pattern promoted in the previous iteration are extracted from the corpus in the current iteration, and vice-versa. As such, each iteration feeds into the next iteration and results in a monotonically increasing set of relation and pattern instances.

3.4.4 Discussion

The feedback strategy in the Espresso system does not add any additional complexity to the system beyond what has already been discussed because it is an inextricable part of Espresso's confidence estimation phase. The same holds true for MBL.

Espresso's feedback strategy is similar to MBL's in the sense that the final set of pattern and relation instances considered and selected in the current iteration depends on information from the previous iteration. Espresso makes decisions about which relations and patterns to retain at the beginning of each iteration by ranking the instances by their confidence estimates. The confidence estimates are affected by the introduction of new instances on every iteration. MBL decides which instances and patterns to extract on the current iteration based on the ones added to the set on the previous iteration; only relations matching a pattern promoted in the previous iteration, and vice-versa, are extracted in the new iteration.

Espresso and MBL fundamentally diverge in their retention policies, however. On each iteration, Espresso only retains k patterns with the highest reliability scores, and k only grows by 1 on each iteration. This means that, unless only one new pattern is extracted per iteration, then there is a probability that patterns retained in previous iterations will be dropped in the current iteration. MBL does not ever drop patterns which it decides to retain. Hence, while the pattern set in each system increases monotonically, Espresso utilizes statistical evidence global to all iterations

to determine which patterns to retain; no pattern is exempt from the possibility of obsoletion. In contrast, MBL utilizes only local statistical information (i.e. from the previous iteration) to determine pattern retention and therefore never discards any patterns which it previously decided to retain.

Furthermore, the feedback strategy of the two systems also determines their respective instance extraction processes. Espresso extracts relation instances from the corpus which match any currently existing patterns. However, as all patterns from previous iterations will not extract new instances, this limits the scope of the extraction phase to all patterns retained from the previous iteration. This will be a subset of all of the new patterns extracted in the previous iteration because Espresso's retention policy may lead to new patterns being discarded before they are ever used to extract relation instances. On the other hand, since MBL does not ever discard patterns, the system uses all of the patterns retained in the previous iteration to extract new instances.

This analysis suggests that Espresso takes a more statistically robust approach whereas MBL has a higher extractive power. These properties correspond to precision and recall, respectively. SRES depends on a single iteration of its extraction pipeline to extract all valid patterns and instances. The effectiveness of these approaches will be discussed in the following section.

3.5 Performance Analysis

This section will describe and compare the performance of each system. For each system, I will indicate on which corpora the system was evaluated and to what other systems it was compared. I will also indicate the system's average precision (AP) and recall (AR), as well as its precision (PR) and recall ratio (RR) to the best-performing comparison system.

3.5.1 Espresso

- **Corpora**: TREC-9[1], CHEM[2]
- **Compared Systems**: RH02 [12], PR04 [15]

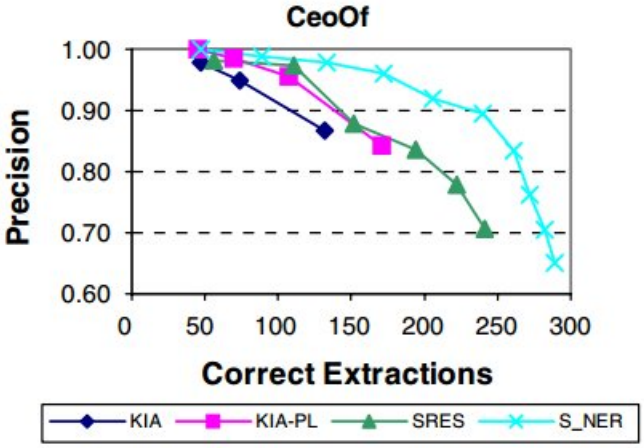
Espresso Performance

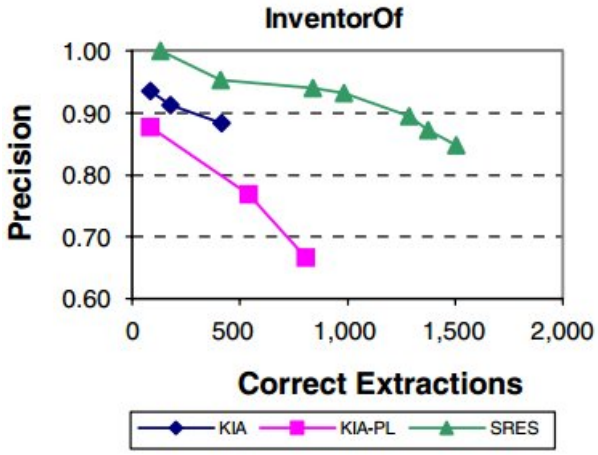
Corpus	AP	PR	AR	RR
TREC-9	0.673	3.11	–	5.05
CHEM	0.772	2.11	–	3.11

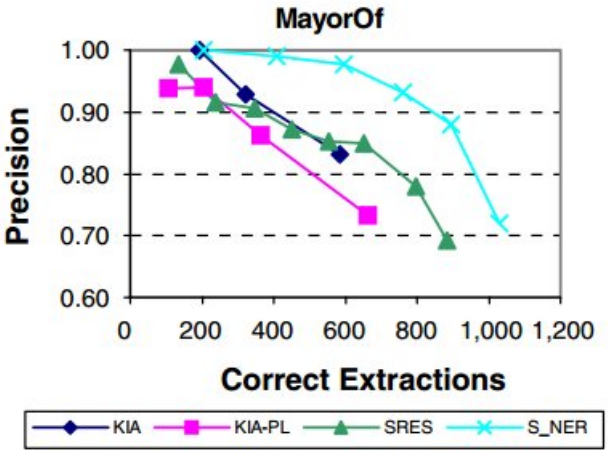
3.5.2 SRES

- **Corpora**: Web
- **Compared Systems**: KnowItAll [16]

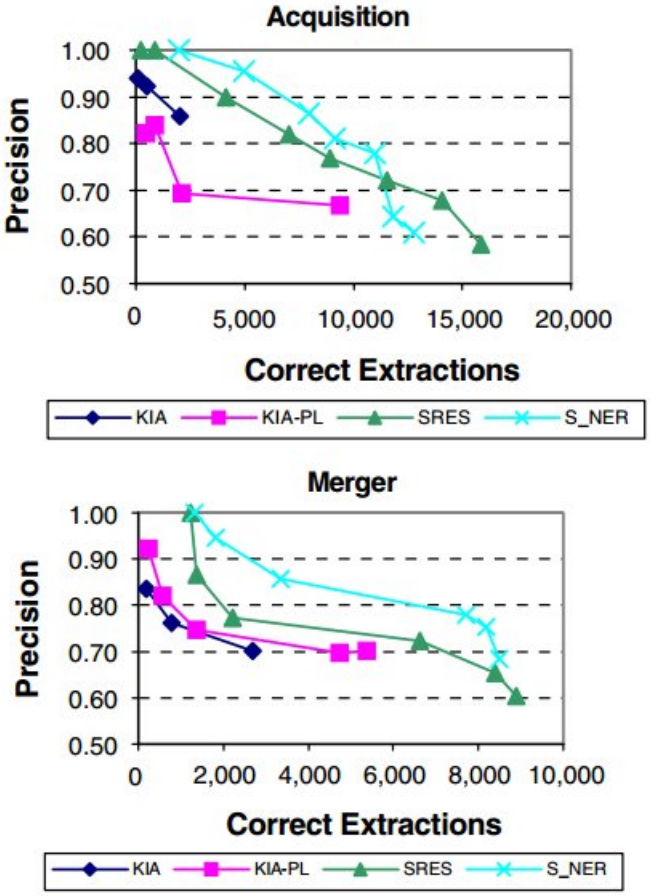
The authors of SRES do not describe explicit performance results, so I will simply include the performance charts presented in the paper.







[1] http://trec.nist.gov/data/qa/t9_qadata.html
[2] http://wps.prenhall.com/esm_brown_chemistry_9/



3.5.3 MBL

- **Corpora**: Web
- **Compared Systems**: UPL [13], CPL, SEAL [14], CSEAL, MBL

MBL Performance

Corpus	AP	PR	AR	RR
Web	0.95	1.04	–	1.89

3.5.4 Discussion

All three systems outperform their respective comparison systems. In terms of raw precision, MBL outperforms the other two systems, achieving an extremely high average precision of 0.95. While the exact average precision of SRES is not clear from the performance graphs, it appears to be approximately 0.8. Espresso places third with an average precision of 0.673 and 0.772 for its two evaluation corpora.

One interesting observation is that SRES and MBL, which construct corpora by crawling the Web, perform significantly more well than Espresso, which uses pre-built corpora. One possibility is that SRES and MBL are simply better-performing systems than Espresso. However, it is also possible that the redundancy of the information found on the Web results in better system performance; both SRES and MBL explicitly mention the use of redundancy in classifying relation instances. The TREC-9 and CHEM corpora are unlikely to be as redundant as the Web. The performance disparity certainly merits further investigation of this issue.

SRES appears to be robust. The performance charts show a relatively smooth precision fall-off as the number of correct extractions (and therefore the overall corpus size) increases. Nonetheless, MBL performances significantly more well than SRES in terms of average precision. The most obvious difference between the two systems is that MBL incorporates a feedback loop and information from previous iterations into its methodology, whereas SRES runs in a single pass. Further analysis is required to determine what is most responsible for the performance in the two systems.

4 Conclusions and Future Work

The analysis presented in this survey pertinent in the context of a bootstrapped, minimally supervised ontology-constructing system which utilizes active learning. Such a system would operate in the following way:

1. Start with a gold-labeled input training set of relations.
2. Select and extract features from the training set.
3. Train a classification model on the training set in order to detect a set of discriminatory features.
4. Use the features to formulate a set of queries to submit to a corpus search engine (e.g., Google for the Web).
5. Perform instance extraction on the results to obtain a set of candidate member relations of the ontology.
6. Classify the candidate relations using the trained classification model.

7. Use a human resource (e.g., a single human expert or Amazon's Mechanical Turk service) in order to verify the assigned labels.

8. Add the new positive labels to the training set.

9. Repeat until a desired termination condition is met.

The general idea behind this system is that higher-level discriminatory features would be detected with repeated iterations, allowing the system to more accurately extract members of the desired ontology. This system is a good example of relation extraction in the context of an iterative feedback loop, and thus merits the investigation of the particular relation extraction systems studied in this paper.

Though I have performed a descriptive analysis of each relation extraction system and a limited comparison of each system's performance, further work is required in order to determine how to best incorporate relation extraction into an active learning system. Most important is to determine whether the performance differences of the three systems are inherent to the methodology which the systems utilize or are simply artifacts of the particular corpora on which the systems were evaluated. Beyond that, experiments which change particular aspects of each system in order to examine impact on performance would also prove beneficial.

It is possible that a particular superposition of these systems would yield better performance than using any of the individual systems in isolation. To best explore the space of possibilities, the various phases of the relation extraction systems could be separated into modular components, and the Configuration Space Exploration (CSE) framework could be applied in order to find the best combination of components [17].

References

[1] Yarowsky, D. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, 1995.*

[2] Zhao, S., & Grishman, R. Extracting Relations with Integrated Information Using Kernel Methods. *In Proceedings of ACL Conference, 2005.*

[3] GuoDong, Z., Jian, S., Zhang, J., & Zhang, M. Exploring various knowledge in relation extraction. *In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, 2005.*

[4] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. Text Classification using String Kernels. *Journal of Machine Learning Research, vol. 2 (pp. 419-444), 2002.*

[5] Bunescu, R., & Mooney, R.J. Subsequence Kernels for Relation Extraction. *In Proceedings of the 19th Conference on Neural Information Processing Systems (NIPS), 2005.*

[6] Zelenko, D., Aone, C., & Richardella, A. Kernel Methods for Relation Extraction. *Journal of Machine Learning Research 3 (pp. 1083-1106), 2003.*

[7] Culotta, A., & Sorensen, J. Dependency tree kernels for relation extraction. *In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, 2004.*

[8] Bunescu, R., & Mooney, R.J. A Shortest Path Dependency Kernel for Relation Extraction. *In Proceedings of the Joint Conference on Human Language Technology / Empirical Methods in Natural Language Processing (HLT/EMNLP), 2005.*

[9] Pantel, P., & Pennacchiotti, M. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. *In Proceedings of Conference on Computational Linguistics / Association for Computational Linguistics, 2006.*

[10] Rozenfeld, B., & Feldman, R. Self-supervised relation extraction from the Web. *Knowledge Information Systems 17(1) (pp. 17-33), 2008.*

[11] Carlson, A., Betteridge, J., Wang, R.C., Hruschka Jr., E.R., & Mitchell, T.M. Coupled Semi-Supervised Learning for Information Extraction. *In Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM), 2010.*

[12] Ravichandran, D., & Hovy, E. Learning Surface Text Patterns for a Question Answering system. *In Proceedings of the 40th American Computational Linguistics conference, 2002.*

[13] Wang, R.C., & Cohen, W.W. Character-Level Analysis of Semi-Structured Documents for Set Expansion. *In Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP), 2009.*

[14] Wang, R.C., & Cohen, W.W. Automatic Set Instance Extraction using the Web. *In Proceedings of Joint Conference of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP), 2009.*

[15] Pantel, P., & Ravichandran, D. Automatically Labeling Semantic Classes. *In Proceedings of Human Language Technology / North American Association for Computational Linguistics (HLT/NAACL), 2004.*

[16] Etzioni, O., Cafarella, M.J., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D.S., & Yates, A. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence 165(1) (pp. 91-134), 2005.*

[17] Yang, Z., Garduno, E., Fang, Y., Maiberg, A., McCormack, C., & Nyberg, E. Building Optimal Information Systems Automatically: Configuration Space Exploration for Biomedical Information Systems. *In Proceedings of ACM International Conference on Information and Knowledge Management (CIKM), 2013.*