Extracting Relations from Unstructured Text

Ryan McDonald
Department of Computer and Information Science
University of Pennsylvania
Levine Hall, 3330 Walnut Street, Philadelphia, PA 19104
ryantm@cis.upenn.edu

Technical Report: MS-CIS-05-06

April 15, 2005

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Scope of Paper . . . 4
1.3 Some Notation . . . 5

2 Snowball: Agichtein and Gravano [5] **5**
2.1 Background . . . 5
2.1.1 Bootstrapping . . . 5
2.1.2 Selectivity vs. Coverage . . . 6
2.1.3 DIPRE . . . 6
2.2 Snowball . . . 7
2.2.1 Initialization . . . 8
2.2.2 Training the Classifier . . . 8
2.2.3 Labeling the Data . . . 10
2.3 Discussion . . . 10

3 Integrated Parsing: Miller *et al.* [7] **11**
3.1 Background . . . 11
3.1.1 Parse Trees . . . 12
3.2 The Collins Parser . . . 12
3.3 Annotating Relations in Parse Trees . . . 14
3.3.1 Some Practical Details . . . 16
3.4 Discussion . . . 16

4 Kernel Methods: Zelenko *et al.* [9] **17**
4.1 Background . . . 17
4.2 Shallow Parsing and Example Creation . . . 17
4.3 Kernel Methods . . . 18
4.3.1 Classification . . . 21
4.4 Discussion . . . 22

5 Discussion and Future Directions **23**
5.1 Quantitative Comparison? . . . 23
5.2 Semi-supervised Methods . . . 23
5.3 Supervised Methods . . . 24
5.4 Semi-supervised vs. Supervised . . . 24
5.5 Non-sentential Relations . . . 24

Acknowledgments

I would like to thank Mitch Marcus for chairing my WPEII committee,
as well as Mark Liberman and Fernando Pereira for sitting on the committee.
Thanks also go to Eugene Agichtein for promptly answering my queries about Snowball
and Nick Montfort for proof-reading a draft of this paper.

1 Introduction

1.1 Motivation

Structured text is all around us. One of the largest sources is the World Wide Web. Google.com, for instance, is currently indexing over 4.5 billion web pages, which include HTML, PowerPoint, PDF, and other forms of structured information. However, the structure of these pages is largely concerned with the visual formatting of data and not with the data's syntactic and semantic properties. Hence, within these structured pages exists a vast amount of unstructured text ready to be mined and exploited in technologies like web-searching, question answering and database generation.

Outside the World Wide Web are other large sources of unstructured electronic text including newsgroups, newspaper corpora (Reuters, the Wall Street Journal, etc.), scientific literature and conference proceedings, governmental transcripts such as parliamentary and court proceedings, email communications, fiction and non-fiction books[1], etc. The list is gigantic.

How can computers help humans make sense of all this data? How can a computer make sense of all this data? Ideally, every piece of information that would ever be needed to answer queries or to sort and search data would be neatly marked in the text with some kind of universally agreed upon standard, such as XML. However, in practice this is rarely the case and most data remains a set of words strung together (albeit in a not so arbitrary way).

This ideal was recently popularized (at least for the world wide web) by Berners-Lee *et al.* [1] in their description of the *Semantic Web*. In the Semantic Web, meaning and language structure are marked up in addition to page format. The major problem facing the Semantic Web is how to mark up billions and billions of pages. The sheer size of the data makes human annotation infeasible. Furthermore, web designers rarely conform to W3C [2] standards when creating pages. Expecting the designers of tomorrow to add an additional layer of markup in future documents is unrealistic.

One course of action would be to have a computer annotate all this electronic data with the structures that are of interest to humans. This, of course, is not trivial. How do we tell or teach a computer to recognize that a piece of text has a semantic property of interest in order for it to make correct annotations?

The focus of this paper will be how to teach computers to recognize *relationships between entities* in unstructured text. An entity is commonly a physical thing such as a person, place, organization, gene or chemical name, but can also be more abstract and encompass things like prices, units, measurements, schedules and even philosophical beliefs. For our purposes an entity is any concept that can be identified in text and related to other entities. In the past decade, a large amount of work has been done on identifying entities from text [3, 4], with current state-of-the-art systems approaching human capabilities [2].

In its simplest form, a relationship is n-ary predicate:

$$r_D(e_1, e_2, \ldots, e_n)$$

[1]It is now possible to search the text of books through Amazon.com.

[2]i.e., approaching inter-annotator agreement levels.

which represents that in document D, the entities, $e_1, \ldots, e_n$ are in relation r to each other. Examples of some binary relations include *location-of*, e.g., li_D(White House, Washington), li_D(Microsoft, Redmond), and *employee-of*, e.g., eo_D(Dennis Ritchie, Lucent), eo_D(Dan Rather, CBS).

It is also possible to define more complex relationships such as genomic variation in genes causing malignancy: gvm_D(gene, variation, malignancy). An even higher order relationship might be one representing sports teams:

$$\textit{baseball-team}_D(\text{name, year, catcher, first-base, second-base, } \ldots \text{, left-field})$$

Such relations would help cancer researchers identify patterns in the behaviour of genes associated with cancer as well as baseball enthusiasts wanting to know who played first base for the Blue Jays in 1992. It seems clear that extracting such information could improve many applications.

Presented here is a partial literature review of extracting relations from unstructured text focusing on three particular papers.

Agichtein and Gravano [5] use a semi-supervised approach similar to the algorithm of Yarowsky [6] for word-sense disambiguation. The algorithm primarily works by using a small set of seed relations to extract meaningful relation contexts. This is then bootstrapped to extract new relation instances and new relation contexts.

Miller *et al.* [7] use an integrated supervised parsing approach. The novelty of their system is to re-annotate natural language parse trees to include relation information at each non-terminal node. Using the re-annotated trees, it is then possible to train a parser (they use the Collins parser [8]) to parse new sentences and extract relation information accordingly.

Zelenko *et al.* [9] also use a supervised learning setting. However, their approach differs in many ways from Miller *et al.* [7]. The system is essentially two staged: the first stage runs a shallow parser on each sentence; the second stage uses a classifier to make yes/no relation decisions about each chunk of the sentence. Each chunk that is classified as having a relation is then returned by the system. Their primary classification mechanism relies on a kernel definition for the shallow parse regions. With this definition, they compare multiple kernel based classifiers, including the voted perceptron [10] and support vector machines [11].

1.2 Scope of Paper

The primary purpose of this critical review is to compare and contrast automatic methods for extracting relations between entities from text. This topic is quite broad and can arguably be broken into three subtasks.

1. **Identifying entities:** As mentioned earlier, this is a well-studied area of research. Extracting entities is not a focus of this paper, and will only be discussed as each section merits.
2. **Annotating relation structures in text:** This is actually an understudied area of research. Unlike entities, relations are often not contiguous and can span multiple sentences or paragraphs within a document. Finding consistent annotation guidelines for such structures can be very difficult. Again, this is not the focus of this paper and it

is assumed that all relations have a structure that is easy to annotate and manipulate. This problem will not be completely ignored. In particular, it will come up in our discussions of the work by Miller *et al.* [7], whose system relies heavily on how the relations are annotated.

3. **Identifying relations:** This is the primary focus of this critical review. Given a set of entities, a set of relations of interest and (possibly) some annotated training examples, how can a computer automatically identify new instances of the sought after relations?

1.3 Some Notation

The standard method used in this paper to indicate a relation is the following:

$$(e_1, e_2, \ldots, e_m)$$

where e_i indicates the ith entity in the relationship. Often, binary relations will be referred to as pairs. For instance, the (*Dan Rather*, *CBS*) pair represents the *employee-of* relation. In the same example, the type of the relation (employee-of) is also left out. This is done when it is clear what relation type is being referred to or if the type is not relevant.

2 Snowball: Agichtein and Gravano [5]

2.1 Background

2.1.1 Bootstrapping

Bootstrapping is a general class of semi-supervised learning algorithms. There are two forms of bootstrapping that garner the most attention in the natural language processing community. Blum and Mitchell's co-training algorithm [12] and the Yarowsky algorithm [6]. At the heart of both algorithms is the notion of a weak learner (or learners) and a large set of unlabeled examples. The algorithm is iterative, using the output of the learner as training data for the next iteration. Ideally, this process will improve performance. Co-training uses two or more learners, each with a separate *view* of the unlabeled data. The output of one is then used as the input for others during the next iteration of training. Yarowsky's algorithm uses just one trainer, taking the highest confidence examples on each iteration as training for the next iteration. Both merit discussion, however the focus here will be on Yarowsky's algorithm as it is the framework primarily deployed by Agichtein and Gravano [5] and other semi-supervised relation extraction approaches.

The basic Yarowsky algorithm is outlined in Figure 1. This definition is very general and does not specify many things, including what classifier to use, how to measure confidence of a labeling or what the convergence criteria is. Yarowsky does provide the details for the problem of word-sense disambiguation, but the framework is general enough so that each problem and data set being studied will warrant their own considerations.

Abney [13] provides a theoretical justification for the Yarowsky algorithm. In particular, he shows that some forms of the algorithm can maximize the likelihood of the unlabeled data. However, the constraints he places on the algorithm are too strong to include Yarowsky's original formulation.

Figure 1: Bootstrapping with the Yarowsky Algorithm. $Conf(D)$ is the set of labellings of data D with confidence greater than some threshold.

```
Input:  A set of seed examples S and a set of unlabeled data D
Algorithm:
  1. T = S
  2. Train a classifier C on T
  3. Label D using C
  4. T = Conf(D) ∪ S
  5. Repeat 2-5 until convergence
  6. Label D using C
```

2.1.2 Selectivity vs. Coverage

The two primary considerations a system design must take into account when using the Yarowsky algorithm are selectivity and coverage. Selectivity refers to our confidence in the classifiers ability to generate precise training examples for future iterations. If the classifier routinely generates false positives, then its accuracy will decrease every iteration, until it becomes of no use whatsoever. This is easily managed by manipulating the classifier to only output positives with extremely high confidence.

However, selectivity must be balanced with coverage. Coverage is the system's ability to generate new (or all) labeled examples. A classifier that is overly selective will not introduce any new examples and the system will terminate without significantly expanding its seed set. Both these issues play a central role in the considerations of Agichtein and Gravano [5].

2.1.3 DIPRE

Before getting into the details of Agichtein and Gravano's Snowball system [5], the system of Brin [14] called DIPRE (Dual Iterative Pattern Relation Extraction) will be discussed. Snowball [5] is essentially an improved version DIPRE and shares much of DIPRE's architecture.

The relation of interest to DIPRE is the (*author*, *book*) relation. However, the system can generalize to any binary relation and with some minor reworking any n-ary relation.

DIPRE starts with a small set of (*author*, *book*) pairs. The system then extracts a tuple for every instance of a (*author*, *book*) seed pair in relative proximity:[3]

[*author*, *book*, *order*, *left*, *middle*, *right*]

where, *order* is 1 if the author string occurs before the book string and 0 otherwise, *left*/*right* are strings containing the 10 characters occurring to the left/right of the match and *middle* the string occurring between the author and book. For example, the tuple extracted for

[3]DIPRE also uses URL information, but it is omitted here since it is not relevant to the discussion.

(*Shakespeare*, *King Lear*) for the string, "*Consider Shakespeare's play King Lear, which tells the tale ...*" would be:

[Shakespeare, King Lear, 1, 'Consider ', ''s play ', ' which tel']

Each tuple extracted is then grouped by matching *order* and *middle*. For each group of tuples, the longest common suffix of the *left* field and the longest common prefix of the *right* field is extracted. Hence, each group induces a pattern:

long-comm-suff(left).AUTHOR.middle.BOOK.long-comm-pref(right)

The above example is for the case when order dictates author before title. Using such a pattern allows the system to extract new examples of (*author*, *book*) pairs. In turn these pairs can generate new patterns.

The primary problem is that some patterns are too easily matched and lead to many false positives. To combat this, DIPRE scores each pattern by |prefix||middle||suffix|, where $|s|$ is the length of string s. Intuitively larger strings are harder to match as they are less common, making these matches more significant. In order to reduce false positives, DIPRE simply throws away all patterns whose score is less than some threshold.

This algorithm is easy to relate to the Yarowsky algorithm. The classifier used by DIPRE is simply a pattern matching system, which is trained by extracting patterns for known (*author*, *book*) pairs. All strings that match at least one of the classifier's patterns are classified as positive and all other strings negative. The (*author*, *book*) pairs in the strings classified as positive are then added to the set of labeled examples to retrain the classifier (i.e., extract more patterns). DIPRE terminates when no new candidate pairs are extracted, or when a human observer decides sufficiently many pairs have been returned.

One of the central insights of DIPRE is that the size of the web allows the use of extremely selective patterns to induce new example pairs of (*author*, *book*). Even with extremely selective patterns, new seed examples will be introduced due to the sheer size of the web. Hence, DIPRE explicitly maintains selectivity by using highly precise patterns and implicitly increases coverage through the size of the unlabeled data set.

2.2 Snowball

Agichtein and Gravano [5] present the system *Snowball* for extracting relations from unstructured text. Snowball shares much in common with DIPRE, including the employment of the Yarowsky bootstrapping framework as well as the use of pattern matching to extract new candidate relations. The relation that Snowball focuses on is the (*organization*, *location*) relation. In order to be consistent, Snowball will be described in terms of the Yarowsky algorithm from Figure 1.

2.2.1 Initialization

The seed set S given as input is a short list of organization/location pairs, (o_k, l_k). Like DIPRE, this list can be very small, and in practice has fewer than ten pairs. Snowball also requires three primary input parameters: τ_{sim}, τ_{sup} and τ_{conf}. The purpose of these parameters will become clear in the following sections.

2.2.2 Training the Classifier

Like DIPRE, the underlying classifier of Snowball is a pattern matching system, in which high confidence patterns are induced by the current set of known pairs, T. Snowball initially defines a confidence measure on pairs in the training set. Initially:

$$Conf((o_k, l_k)) = 1.0 \ \forall (o_k, l_k) \in S$$

Given a set of known related pairs:

$$T = \{(o_k, l_k)\}_{k=1}^{|T|}$$

Snowball extracts a tuple for every string in which a known location and organization pair, $(o_k, l_k) \in T$, are close to one another: $[l, e_1, m, e_2, r]$. Where $e_1, e_2 \in \{loc, org\}$ & $e_1 \neq e_2$. m is a feature vector that represents the tokenized terms that occur between the identified pair. Similarly l and r are also feature vectors representing the tokenized terms occurring to the left or right of the pair up to some limit on the number of terms. For example, if (*Microsoft*, *Redmond*) is a known pair, then for the string "*... while at Microsoft's headquarters in Redmond, there were ...*" the extracted tuple would be:

[{<v_1, while>, <v_2, at>}, org, {<v_2, 's>, <v_4, headquarters>, <v_5, in>}, loc, {<v_6, ,>, <v_7, there>}]

This is for the case when the limit on the left and right feature vectors is 2.

Several values, $v_i \in \mathbb{R}$, have been introduced. These are the corresponding weights for each term. A weight for a term is calculated by the normalized frequency of that term in the left, middle or right context. Hence a term can have a different weight when it occurs in the m feature vector then it might have in the l feature vector. For example the weight for the term *at* in the left context would be:

$$weight(\text{at}, l) = \frac{count(\text{at} \in l)}{\sqrt{\sum_w (count(w \in l))^2}}$$

These weights are updated every iteration since every iteration will introduce more tuples. Finally the weights are adjusted to allow for different contexts to be assigned different levels of importance:

$$weight(\text{at}, l) = \lambda_l \cdot weight(\text{at}, l)$$

λ_l, λ_m and λ_r are all input parameters.

Agichtein and Gravano define a similarity function over extracted tuples:

$$Match(tup_i, tup_j) = \begin{cases} (l_i \cdot l_j) + (m_i \cdot m_j) + (r_i \cdot r_j) & \text{if } e_{1,i} = e_{1,j} \ \& \ t_{2,i} = t_{2,j} \\ 0 \text{ otherwise.} & \end{cases}$$

$$tup_i = [l_i, \ e_{1,i}, \ m_i, \ e_{2,i}, \ r_i]$$

Clearly tuples that share common terms in their feature vectors are going to have higher similarity over those that do not. Also note that this is a much softer matching criteria than used by DIPRE. Instead of having hard textual matches, Snowball's match metric allows for slight variations in tokens or punctuation. This is primarily a coverage metric that helps to identify pairs that exact matching systems might miss.

Snowball then induces patterns in two steps. The first step is to cluster all the tuples into a set of groups, $G = \{g_1, \ldots, g_m\}$, $g_k = \{tup_1^k, \ldots, tup_n^k\}$, using the similarity function $Match$. In the second step, each group, $g_k \in G$ induces a tuple pattern:

$$p_{g_k} = [l_C, e_1, m_C, e_2, r_C]$$

where l_C, m_C and r_C are the centroids of all the left, right and middle feature vectors for the tuples in the group. By the definition of the similarity metric, $Match$, every tuple belonging to the same group will have identical values for e_1 and e_2.

As noted earlier, a major concern of Yarowsky style bootstrapping is the selectivity of the classifier. Following the garbage-in-garbage-out principle of classifiers, it would be wise to make sure that the classifier only spits out precise information. DIPRE handles this by favoring long patterns, thus making matches less likely.

Snowball handles selectivity by first removing all patterns that were induced by less than τ_{sup} extracted tuples (i.e., the group that induced the pattern only contained a small number of tuples). A confidence score is then assigned to each pattern. On every iteration Snowball measures, for each pattern p_{g_k}, the number of positive and negative pairs resulting from the application of that pattern. A pair (o_k, l_k) is considered positive if, on the current set of pairs used to induce the patterns, T, there exists a pair $(o_k, l'_k) \in T$ with a high confidence value, $Conf((o_k, l'_k))$, and $l_k = l'_k$. (o_k, l_k) is considered negative if if there is a high confidence pair with $l_k \neq l'_k$ and neutral otherwise. Using this, the confidence of a pattern, p_{g_k}, is defined as:[4]

$$Conf(p_{g_k}) = \frac{\text{num-pos}(p_{g_k})}{\text{num-pos}(p_{g_k}) + \text{num-neg}(p_{g_k})}$$

In other words, if Snowball has confidence in the pairs extracted by previous iterations of the system, and this pattern generates lots of those pairs and few contradicting pairs, then the system will have confidence in those patterns.

Now that a confidence metric has been introduced, the simple approach would be to only use patterns with highest confidence to introduce new pairs for the next iteration, which is the method of DIPRE. However, Snowball takes a different approach. Instead Snowball uses the confidence measure of patterns to recalculate the confidence of the pairs that the induced patterns extract. Only those pairs with highest confidence are kept for the next iteration.

2.2.3 Labeling the Data

To extract new pairs, Snowball first runs a named-entity tagger over the data to identify all the location and organization entities within the documents. For each organization/location

[4]Note that the $Conf$ function is overloaded. There are versions for both patterns and as well related pairs.

pair, (o, l) that are within the same sentence, the system extracts a tuple, $tup_{(o,l)}$ in the same manner as in the previous section. Hence, a pair that occurs many times will have a set of tuples associated with it, $tup^{(j)}_{(o,l)}$. This tuple is then compared to all the induced patterns that were previously extracted and introduced to the classifier.

For each candidate pair, (o, l), the system records which patterns match the pair with a similarity greater than τ_{sim}, as well as what the similarity value is. More concretely, for (o, l), the system stores a set:

$$M = \{< p_{g_k}, Match(tup^{(j)}_{(o,l)}, p_{g_k}) > \mid \forall p_{g_k},\ tup^{(j)}_{(o,l)}\ s.t.\ Match(tup^{(j)}_{(o,l)}, p_{g_k}) > \tau_{sim}\}$$

Let $M_i[0]$ be the pattern involved in the ith entry of M and $M_i[1]$ be the similarity score causing this entry.

Snowball defines the confidence of a pair, (o_k, l_k) as:

$$Conf((o_k, l_k)) = 1 - \prod_{i=0}^{|M|}(1 - (Conf(M_i[0]) \cdot M_i[1]))$$

This value is high when the pair (o_k, l_k), is matched by many tuples with high similarity to patterns that the system is confident in.

The seed set for the next iteration is set to the original seed set, plus the candidate pairs with highest confidence (confidence greater than τ_{conf}).

2.3 Discussion

The primary advantage of Snowball and other related semi-supervised training systems is that they require little to no human annotation. Later, the work of Miller *et al.* [7] and Zelenko *et al.* [9] will be discussed, both of which rely heavily on a large set of manually annotated examples.

Having said this, Snowball relies on the use of a named-entity recognizer when matching patterns. Current state-of-the-art entity recognizers are largely supervised systems [3, 4], requiring annotated training data. Identifying organizations and locations is not problematic due to the availability of data annotated with these entities. However, this is not the case for all entities and it may be difficult to extend Snowball to incorporate other relation types.

Furthermore, Snowball relies on an intrinsic property of organizations and locations - that every organization has its headquarters in only one location - when calculating the confidence score of a pattern. This property does not hold for all relations. For instance, in the *author-of* relation, one author can be associated with many books and one book with many authors. Even organizations can have multiple headquarters in different parts of the world.

One major disadvantage of Snowball is its reliance on a large number of input parameters (I counted nine in total). The definition of most of these parameters is clear, but there is no guarantee that good values on one set of data will translate to good values on all sets of data. However, these parameters do provide a method for which users can balance their requirements of the system.

A simple refinement to Snowball would be to replace the pattern matching method of extracting new seed pairs with a more sophisticated classifier. For instance, Yarowsky [6]

uses a decision list algorithm to classify word-sense. Similarly, one could extract every organization/location pairs in relative proximity and create a feature vector of their surrounding context. The feature vectors for known pairs can be used to train the classifier. In fact, this would allow for the training of almost any classifier, including current state-of-the-art classifiers like maximum entropy [15] and support-vector-machines [11]. These classifiers usually define a probability (or some similar notion) to each classification, which would naturally substitute for confidence in order to maintain selectivity. Supervised classifiers are also usually quite principled, either minimizing error or maximizing likelihood of the labeled data.

At the beginning of this section an alternative semi-supervised bootstrapping algorithm was mentioned: co-training. Recent studies on natural language parsing have shown that co-training actually outperforms Yarowsky style bootstrapping in empirical tests [16]. Though the domains are somewhat different, this does provide evidence for improved semi-supervised approaches.

Yangarber [17] proposes semi-supervised relation extraction using a co-training like algorithm called counter-training. Unlike co-training, the multiple classifiers provide negative and not positive information to the other classifiers. This negative information will eventually cause all the classifiers, save one, to stop acquiring patterns. Yangarber argues that each classifier learns a specific subset of patterns with high precision, since low precision patterns may intrude on another classifiers domain and get negative information from that classifier. This way it is possible to maintain selectivity through the precision of each classifier and coverage through the use of multiple classifiers.

The main problem with Yarowsky style bootstrapping algorithms according to Yangarber [17] is that the patterns that the system extracts degrade with every iteration since ultimately some errors will be introduced to the system. With co-training or counter-training algorithms, the multiple classifiers play a role in preventing this from happening by constraining each other through the different views of the data.

3 Integrated Parsing: Miller *et al.* [7]

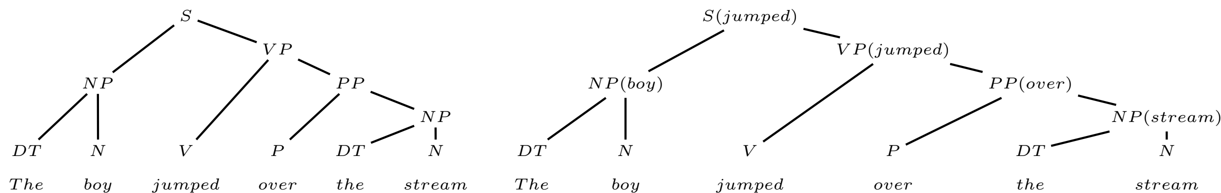
3.1 Background

This sections moves away from the analysis of semi-supervised methods into the domain of fully-supervised methods. In particular, the focus will be on the work of Miller *et al.* [7], which use an integrated parsing approach to extracting relations.

The primary motivation of Miller *et al.* is the observation that most relation extraction systems are pipelined. Usually entity tagging and other syntactic tagging such as part-of-speech and parse tree generation are done beforehand using separately trained models. The output of these models are then used as input to the relation extraction model. In the last section, the system of Agichtein and Gravano [5] indeed use an entity tagger's output as its gold standard. In fact, Snowball can be seen as a repeating pipeline, in which output from one stage is taken as gold standard for the next. Hence, errors at one stage propagate to errors in future stages.

Agichtein and Gravano are acutely aware of this. That is why selectivity is a central issue

Figure 2: An example parse tree. The version on the right is has been lexicalized.



when devising semi-supervised methods. However, Miller *et al.* take a different approach. Their system makes all relation, entity and syntax decisions at once using a generative probability model. They create this model by encoding all decisions into natural language parse trees. Once this is done, then the model of interest becomes $P(T, S)$, where T is a parse tree and S the input sentence.

3.1.1 Parse Trees

The data structure primarily employed by Miller *et al.* [7] are natural language parse trees. Each node in a parse tree is labeled to represent the linguistic entity in which that node subsumes, i.e., noun phrase by NP, verb phrase by VP and sentence by S. Parse trees can be broken down into the root node (usually S), internal nodes, preterminal nodes and terminal nodes. The terminal nodes, or the leafs of the tree, are the words of the sentence. The preterminal nodes represent the part-of-speech of each word, such as noun, preposition and determiner. Figure 2 shows an example parse tree.

A lexicalized parse tree is one in which a *head* word is assigned to each internal node of the tree. Intuitively the head word is the most representative word of the phrase that a internal node subsumes. This is typically the verb of a verb phrase or the right most noun in a noun phrase. Collins [18], provides a deterministic algorithm for lexicalizing a parse tree that is widely used. Figure 2 also shows a lexicalized parse tree.

3.2 The Collins Parser

Recently, due in large part to the availability of a large tagged corpus [19], much work has been done in the area of statistical supervised automatic parsing techniques [20, 18]. One of the most widely used parsing models is the lexicalized generative model of Collins [18]. The model presented here is the variation of Collins model used by Miller *et al.*

Collins model works by generating information from the head outward. The basic generative process is outlined in Figure 3. It can be seen as a depth-first model that starts with some non-terminal, generates head information for that non-terminal, and then begins to generate left and right modifier information. In order to begin the process, a head word and POS tag must be generated for the sentence from the start distribution $P(h_S, t_S|START)$. Modifier generation is terminated when the model generates the *null* modifier.

Figure 3: Modified Collins parsing model used by Miller *et al.*

```
GenerativeParse(N, w_N, t_N)
  1. generate head child non-terminal C from P(C|N)
  2. while (generate left modifier non-terminal M_{l,i} ≠ null from P(M_{l,i}|N,C,w_N,M_{l,i-1}))
  3.     generate head POS for modifier t_{M_{l,i}} from P(t_{M_{l,i}}|M_{l,i},w_N,t_N)
  4.     generate head word for modifier w_{M_{l,i}} from P(w_{M_{l,i}}|M_{l,i},t_{M_{l,i}},w_N,t_N)
  5.     generate head word feature for modifier f_{M_{l,i}} from P(f_{M_{l,i}}|M_{l,i},t_{M_{l,i}},w_N,t_N,known(w_{M_{l,i}}))
  6.     GenerativeParse(M_{l,i},w_{M_{l,i}},t_{M_{l,i}})
  7. Repeat 2-6 for right modifiers
```

$$known(w) = true \text{ if } w \text{ is known word.}$$

The model used by Miller *et al.* differs from the original Collins model in many significant ways. Primarily, the Miller *et al.* model uses a first order Markov assumption when generating new modifiers, whereas Collins uses a 0th order model. Miller *et al.* also generates word features for unknown words, such as capitalization. This is most likely a way to improve entity recognition, since orthographic features are often very indicative of a tokens entity type. Miller *et al.* also make no mention of complement distinction or sub-categorization generation and it is unclear what part, if any, these are considered in the model. Finally, there is no mention of distance or verb crossing statistics in the distributions.

Like Collins, Miller *et al.* use smoothed maximum likelihood estimates to calculate the required probability distributions. For example, the probability of generating a word given its history is defined as:

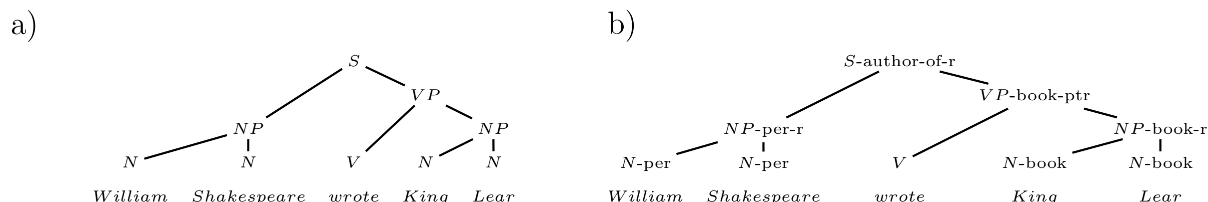
$$P(w_{M_{l,i}}|M_{l,i},t_{M_{l,i}},w_N,t_N) =$$
$$\lambda_1 P(w_{M_{l,i}}|M_{l,i},t_{M_{l,i}},w_N) + \lambda_2 P(w_{M_{l,i}}|M_{l,i},t_{M_{l,i}},t_N) + \lambda_3 P(w_{M_{l,i}}|M_{l,i},t_{M_{l,i}}) + \lambda_4 P(w_{M_{l,i}}|t_{M_{l,i}})$$

The purpose of these smoothed estimates is too account for sparseness in counts for distributions with a lot of history by backing off to less sparse estimates. The parameters λ_i can be calculated in any number of ways. The only constraint is that $\sum_i \lambda_i = 1.0$.

In order to find the best parse given an input sentence, the entire search space must be explored. Though this space is exponential in size, it is still possible to search it in polynomial time since all calculations are over local parent/child subtrees. This allows for a modified version of CKY [21] to be used. The complexity is still high at $O(n^5)$ for lexicalized trees, making the use of pruning necessary to limit the size of the search to those chart items with highest score.[5] The best parse can be retrieved by following back pointers from the constituent with highest probability that is headed by the start symbol and spans the entire sentence.

[5]The score is the probability of a constituent generated by the model multiplied by the prior probability of the non-terminal heading that constituent [22].

Figure 4: a) A parse tree with entity annotations but no relation annotations. In this tree *William Shakespeare* is related to *King Lear*, through the *author-of* relation. Lexical information left of for simplicity. b) The same parse-tree but with relation information included.



3.3 Annotating Relations in Parse Trees

A fundamental insight of Miller *et al.* [7] is the realization that by encoding relation and entity information into a parse-tree's non-terminals, results in the ability to train a state-of-the-art parser to extract relations. No additional models are necessary for relations or entities since they are encoded in the resulting parse trees.

For entities, it is fairly straight forward to encode them in a tree provided that all parse trees obey entity boundaries. All that is required is to find the lowest most node in the tree that covers all and only the tokens of an entity and augment the label of the tree to include information about the entity. Miller *et al.* label this lowest most node as a reportable type. For example, if the node is a noun phrase and covers an organization entity, then the node would be labeled *NP-org-r*. All nodes in the tree rooted at this node would also be augmented to store entity information, except that these nodes would not be reportable (i.e., *N-org*). This is a key distinction in the annotation guidelines. Only those nodes marked up as reportable represent complete entity or relational information.

Encoding relations is a little more complicated. To demonstrate the procedure, the two instances identified by Miller *et al.* in which a relation might occur in a sentence will be examined. The first is when two non-overlapping and non-modifying entities are related in a tree. Consider the example in Figure 4. In this sentence there are two related entities, *William Shakespeare* and *King Lear*, which are related through the *author-of* relation. To annotate this relation, the system finds the lowest-most node that subsumes both entities, in this case the start node S. This node is then augmented to indicate the *author-of* relation type. The *book* entity is not a direct child of the relation tag, so pointers are added to make it possible to traverse the tree and find the appropriate argument. This would be necessary in the case of argument ambiguity, e.g., *William Shakespeare wrote King Lear and not The Inferno.* Miller *et al.* also include nodes to distinguish the arguments of the relation. Most likely this is necessary when one node subsumes multiple relations of the same type (e.g., *William Shakespeare wrote King Lear, Richard III and Othello.*), otherwise the pointer nodes should be sufficient as is the case in Figure 4.

Figure 5: A parse tree with entity annotations but no relation annotations. This is the case when one entity in the relation modifies the other.

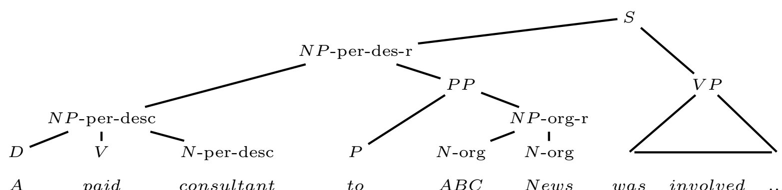
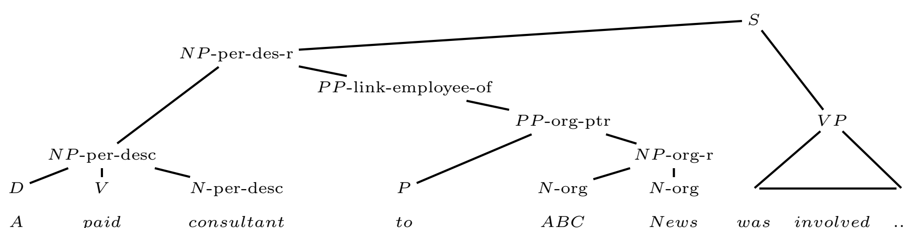


Figure 6: Parse tree from Figure 5 with relations annotated.



The second case handled by Miller *et al.* is when one entity is a syntactic modifier of another. One example given is *A paid consultant to ABC News*, in which *ABC News* is an organization and the whole string a person description, with both entities in an *employee-of* relation. This is the more difficult case when the modifier entity is actually part of the entity being modified. It can be argued whether this is the correct annotation in this case. One possible annotation would have *a paid consultant* as a person description. However, it must be assumed that such relations are possible since they existed in the guidelines for the data used by Miller *et al.* Consider the example given in Figure 5. By the previous guidelines one can find the lowest most node that subsumes both entities involved in the *employee-of* relation. However, this node is actually the reportable person description entity. The simple approach would be to simply add a relation annotation to this node. Miller *et al.* do not take this route and supply no reason; however, one can imagine that further annotation of nodes that have already been labeled as entities may result in sparse data issues. Instead, Miller *et al.* insert a *link* node directly below the topmost node and the child of that node that subsumes the second entity in the relation (the organization in this case). This node is then labeled with the *employee-of* relation and receives the same syntactic category as the child node. Finally pointers to the entities are included so that it is possible to find the reportable entities directly from the node labeled with the relation. The modified tree from Figure 5 can be seen in Figure 6.

These guidelines for relation annotation seem to be incomplete. An example given earlier was, *William Shakespeare wrote King Lear, Richard III and Othello.* In this case one node

subsumes three different binary author relations and it is not specified what the correct tree augmentation should be. Also not specified is how to handle crossing relations, such as *Shakespeare and Dante wrote King Lear and The Inferno respectively.* A sound and complete guideline for annotating all relations in parse trees is a daunting task and beyond the scope of this review. One can only assume that since Miller *et al.* do not discuss any difficulties, that their annotation guidelines were sufficient for their data.

The fact the the annotation guidelines are a times unclear should not detract from the method proposed. The key insight was that augmenting trees to include relation information, allows for the easy creation of a model which neatly incorporates all syntactic and semantic decisions. The choice of how to actually encode the trees is a separate issue. It is not one that should be ignored, and the fact that Miller *et al.* do not provide any motivation for their choice of annotations is disappointing.

3.3.1 Some Practical Details

In order to create a large set of labeled examples, Miller *et al.* run the Collins parser trained on the Penn Treebank [19] over their training data. The model is run so that the parser only outputs trees that obey the predefined entity bracketing. The trees are then annotated with entity and relation information. Finally, it is possible to retrain the Collins parser on the augmented trees in order to tag new sentences.

3.4 Discussion

The intuition behind the integrated parsing approach seems sound. Every entity, relation, POS, and parse tree decision is related and they should all be made at the same time. In particular information about relations, entities and tree structure are highly correlated. For instance, if there is evidence that the tree structure should have a verb phrase headed by *is* that is modified by a noun phrase headed by *author* that contains a prepositional *of* phrase, then this is strong evidence that there needs to be an author entity to the left and an *author-of* relation tagged in the tree. Similarly, there are probably few syntactic variants for which this relation may be embedded. Hence, author and book entity information provides evidence for one of these syntactic structures.

This last point raises a possible criticism of the model. Collin's parsing model only considers local pairwise dependencies with very little history (relative to the entire tree). A major advantage of using parse information to influence relations and vice versa is that through parse trees, long-range structural information can give rise to relations between distant entities. However, since parsing models are constrained to be local (due to complexity issues), it is hard to see how it is possible to improve the accuracy of long range dependencies. A possible solution to this problem is the recent use of re-ranking methods for parsing models [23]. Re-rank models take the k-best candidates from a (usually) generative parsing model and apply a discriminative classifier over them to pick the best candidate. The primary advantage is that these models can contain features over the entire structure of the tree, since the search is constrained to only consider k different parses. This could allow for the use of features over long range dependencies, which could significantly improve relation identification.

The model of Miller *et al.* is also sentential model and on the surface appears to only manage simple relations. Complex relations can exist over multiple sentences making such relations difficult to extract from a sentential system. Furthermore, it is not completely clear how to extend the tree annotation procedure to include relations containing arbitrarily many entities or multiple relations that are both represented by the same lowermost node in a tree. Most of these annotation problems can most likely be fixed by increasing the amount of markup on internal nodes or by inserting new nodes into the trees, but this will eventually lead to sparser probability estimates since the set of labels will undoubtedly increase dramatically.

4 Kernel Methods: Zelenko *et al.* [9]

4.1 Background

As noted in the last section, one of the primary disadvantages of the Miller *et al.* parser is its inability to incorporate long-range features into relation decisions. Another possible drawback is the use of a generative parse model since generative models cannot easily represent a rich set of dependent features in a computationally tractable manner[6].

The model of Zelenko *et al.* [9] is designed to combat both these problems. By using the output of a shallow parser as its gold standard, it is not constrained to create the parse and can instead consider non-local dependencies similar to parse re-ranking. For each shallow parse, the model generates all possible relation instantiations and makes straightforward yes/no classifications on each instantiation to determine what relations, if any, a shallow parse may contain. This allows for the use of powerful discriminative classification techniques, which can easily handle millions of highly dependent features. Their discriminative methods of choice are kernel-based methods such as SVMs [11] or the voted perceptron [10].

4.2 Shallow Parsing and Example Creation

Unlike Miller *et al.*, Zelenko *et al.* use shallow parses and not full parses to encode relations. A shallow parse is like a full parse, except it only aims to identify the basic surface level components of a sentence, such as noun phrases and entities. The shallow parser used by Zelenko *et al.* identifies noun-phrases, people, organizations and locations as well as the part-of-speech tags of those words that occur outside noun-phrases or within noun-phrases when there are non-noun words. Figure 7 shows an example.

Once the shallow parse regions of a sentence have been established, the primary question asked is whether a subtree is an example of the relation of interest. Zelenko *et al.* are restricted to those relations that consist of people, organizations or locations. However, the method generalizes as long as there is access to shallow parses that identify the entity components of a relation.

Assuming there is a large set of labeled data, it is possible to create a set of positive and negative examples for classification. For example, say there was interest in the *employee-of* relation. First a sentence is parsed with the shallow parser. Then, for every

[6]It should be pointed out that for parsing, generative models still represent the state-of-the-art.

Figure 7: Example of a shallow parse structure used by Zelenko *et al.* [9]

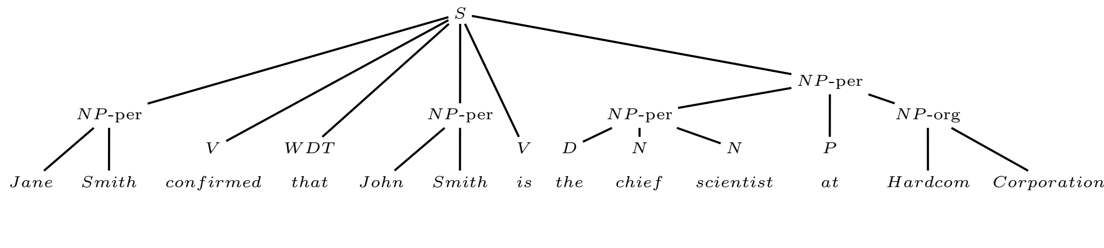
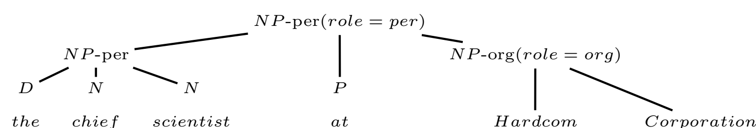
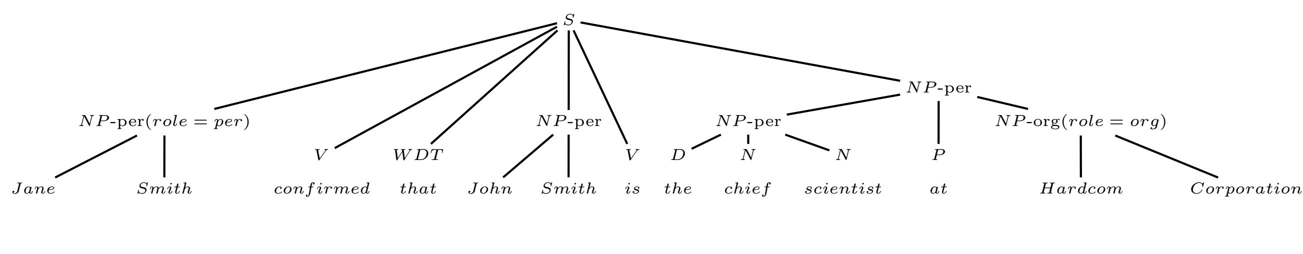


Figure 8: Extracted positive and negative examples

Positive Example:



Negative example:



person/organization pair in the tree, the lowest common node subsuming both entities is found and the subtree rooted at that node extracted. The entity nodes are labeled with a role (e.g., person or organization) in the relation. If those entities are known to be related, then the subtree is given a positive classification and negative otherwise.

Considering the example in Figure 7, it is possible to construct multiple positive examples (when the person role is 'John Smith', 'the chief scientist' or 'the chief scientist at Hardcom Corporation' and the organization role is 'Hardcom Corporation') and one negative example (when the person role is 'Jane Smith' and the organization role 'Hardcom Corporation'). Figure 8 displays two extracted labeled examples.

4.3 Kernel Methods

Having extracted various positive and negative examples it is fairly straightforward to create a classifier to identify sub-trees containing the relation of interest. For instance, one could extract a feature vector from the tree and train a linear-regression, perceptron or naive Bayes model. The problem would then be reduced to finding appropriate features for this problem.

Another approach, which has gained popularity recently [11], is to use kernel-based methods. A kernel is a similarity function, $K(x, y) : X \times X \rightarrow \mathbb{R}$, over input pairs (in our case

subtrees) that must be symmetric and positive-definite.

The primary property of kernels, often called the *kernel trick*, is that every kernel implicitly represents the dot product between two inputs in a high dimensional feature space. This follows from Mercer's theorem [24] that states all kernel functions can be written as:

$$K(x, y) = \sum_i \lambda_i \phi_i(x) \phi_i(y)$$

where $\phi_i(x)$ is the ith eigenvalue of x.

The kernel trick is very important. It allows for the substitution of a kernel in any learning method which can be reformulated so that all calculations on input feature vectors are pairwise dot-products. A surprisingly large number of learning paradigms can be reformulated into this form, including the perceptron algorithm and support vector machines, both of which Zelenko *et al.* consider.

There are many motivations for using kernel-based methods as opposed to explicit feature representations. Kernels can make feature vector calculations in large or even infinite spaces usually on the order of the size of the input and not on the size of the corresponding feature space[7]. Also, all kernel calculations need be to be made only once, when computing the Gram matrix $K_{x,y} = K(x, y)$. Feature spaces also require large maps from subsets of the input to dimensions in the space, which can be both difficult to manage and difficult to store in memory if the number of dimensions grows too large. Kernels, on the other hand, simply require a function definition and space on the order of $O(T^2)$ to store the Gram matrix, where T is the size of the training data.

Before the definition of a relation subtree kernel is given, some notation must be specified. Specifically, each node in a subtree, N, has both a role, $N.role \in \{none, per, org, etc.\}$, a type, $N.type \in \{NP, NP\text{-per}, V, etc.\}$ and a substring of text in which that node subsumes $N.text$. Zelenko *et al.* define the following kernel on two subtrees rooted at N_1 and N_2:

$$K(N_1, N_2) = \begin{cases} 0 & \text{if } t(N_1, N_2) = 0 \\ k(N_1, N_2) + K_c(N_1, N_2) & \text{otherwise.} \end{cases}$$

$$t(N_1, N_2) = \begin{cases} 1 & \text{if } N_1.role = N_2.role \ \&\ N_1.type = N_2.type \\ 0 & \text{otherwise.} \end{cases}$$

$$k(N_1, N_2) = \begin{cases} 1 & \text{if } N_1.text = N_2.text \\ 0 & \text{otherwise.} \end{cases}$$

To define K_c, a definition for subsequences of child nodes for a node N is needed. Say a node N has ordered children $C = \{C_1, C_2, \ldots, C_m\}$ (from left to right). Define $C_S \subseteq C$ as any subset of C such that the order of the nodes in C_S obey the same order as in C (e.g., C_4 cannot occur before C_2 in C_S). Let $C_S[i]$ represent the ith member of C_S. Also, define a function $D(C_S)$ as the difference in index between the rightmost child and leftmost child in the subset[8]. Then K_c is defined as:

[7] Usually this problem is overcome by sparse representations.

[8] e.g., $D(\{C_2, C_4, C_9\}) = 9 - 2 = 7$.

$$K_c(N_1, N_2) = \sum_{C_{S_1}, C_{S_2}, |C_{S_1}|=|C_{S_2}|} \lambda^{D(C_{S_1})} \lambda^{D(C_{S_2})} \sum_{i=1}^{|C_{S_1}|} K(C_{S_1}[i], C_{S_2}[i]) \prod_{j=1}^{|C_{S_1}|} t(C_{S_1}[j], C_{S_2}[j])$$

This is a rather lengthy definition of a kernel, so it will broken down piece by piece for a better understanding. The simple components of the kernel are the functions $t(N_1, N_2)$, which is an indicator function checking that two nodes have the same syntactic type and semantic role, and $k(N_1, N_2)$, which is an indicator function checking that two nodes subsume the same textual string. By examining the kernel function $K(x, y)$ it is clear that if two subtrees do not have roots with the same type and role, then they have a value of 0.

The key computation of the kernel is the function $K_c(x, y)$. This function sums over all equal length child subsequences of two nodes N_1 and N_2 adding the value of the recursive kernel calls to the aligned children in the subsequences. The product term in this calculation requires that every aligned child node in the subsequence has the same type and role for a subsequences contribution to be added to the sum. Finally, each subsequence contribution is weighted by the (user-defined) parameter λ, $0 < \lambda < 1$, raised to the power of the distance between the rightmost and leftmost child in the subsequence. This results in spread-out matching subsequences being penalized.

What does the kernel value actually represent? First of all it is the sum over all common partial-trees between N_1 and N_2 that can be created only by removing internal subtrees. Each node in the common partial-tree is weighted by the λ values plus 1 if the node subsumes the same text in the original trees. The weight of each tree is thus the sum over the weight of all the nodes in the tree.

Zelenko *et al.* provide an $O(mn^3)$ algorithm for calculating $K_c(N_1, N_2)$ where m and n are the number of children of N_1 and N_2, $m \geq n$ and K_c has already been calculated for all children. For two trees N_1 and N_2, it is then possible to compute K_c, for every pair of matching nodes bottom up starting with the nodes that span the smallest amount of text. Assuming in the worst-case that all internal nodes in the two trees are matching and that there are k internal nodes in the largest subtree, then the calculation is $O(k^2mn^3)$ in total, where m and n are the longest sequences of children for some node in the two respective trees.[9]

Finally, Zelenko *et al.* provide a proof that $K(x, y)$ is actually a kernel. The proof relies on the fact that the base functions k and t are trivially kernels as well as the property that kernels are closed under multiplication and addition. Using structural induction on the subtrees with their leafs as the base case it can be shown that K_c is a kernel. It then follows that the whole function K is a kernel.

4.3.1 Classification

As noted earlier, every kernel implicitly represents the dot product of the two input examples in some high dimensional space. Therefore, any learning algorithm that can be reformulated

[9]Zelenko *et al.* present two formulations of their kernel. This first is when child subsequences may only contain contiguous children and the second is when the subsequences can be sparse. The first formulation leads to better complexity for the kernel calculation, but the latter leads to significantly better performance. I assume the second formulation throughout this review.

Figure 9: Perceptron algorithm. a) the algorithm in its more familiar form, b) the reformulated kernel version of the algorithm.

a)

Input: training instances $\{(y_i, \mathtt{x}_i)\}_{i=1}^T$

1. $\mathtt{w} \leftarrow 0$
2. for $i : 1, \ldots, T$
   (a) if $y_i(\mathtt{w} \cdot \Phi(\mathtt{x}_i)) \leq 0$
   (b) $\mathtt{w} \leftarrow \mathtt{w} + y_i \Phi(\mathtt{x}_i)$
3. end for
4. repeat 2-3 until convergence

b)

Input: training instances $\{(y_i, \mathtt{x}_i)\}_{i=1}^T$

1. $\alpha \leftarrow 0$
2. for $i : 1, \ldots, T$
   (a) if $y_i(\sum_j \alpha_j y_j (\Phi(x_j) \cdot \Phi(x_i))) \leq 0$
   (b) $\alpha_i \leftarrow \alpha_i + 1$
3. end for
4. repeat 2-3 until convergence

so that each input example is only used in dot product calculations with other input examples can be considered a kernel method, since it is always possible to substitute a kernel calculation for a dot product calculation.

Possibly the simplest kernel method is the perceptron algorithm. This is presented in Figure 9. By observing the update step of the perceptron algorithm, it can be easily shown that each training pair instance $(y_i, \mathbf{x}_i)$ is added α_i times to the final weight vector $\mathbf{w}$, where α_i is a natural number. Hence, the final weight vector is equal to $\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$. Furthermore it is possible to calculate α_i by adding 1 at each iteration in which $\mathbf{x}_i$ is misclassified and it is easy enough to show that the weight vector at any stage can be calculated by, $\mathbf{w} = \sum_j \alpha_j y_j \Phi(\mathbf{x}_j)$, thus:

$$\mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j \alpha_j y_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) = \sum_j \alpha_j y_j (\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i))$$

The result is that the perceptron algorithm can be rewritten as b) in Figure 9.

In this reformulated perceptron algorithm, input is only used during the dot product calculation of the feature vectors. Hence, a kernel function may be substituted for that dot product. It is not difficult to show that this result also generalizes for the voted perceptron.

Zelenko *et al.* experiment with both the Voted Perceptron and support vector machines (SVMs). SVMs are similar to the perceptron in that they find a separating hyperplane (when the data is separable), except that SVMs guarantee that the hyperplane returned will be that which maximizes margin. Briefly, maximizing margin will result in a hyperplane that is as far away as possible from the positive and negative examples closest to the decision boundary. The motivation is that max margin will lead to improved regularization since it fits equally well to both the positive and negative inputs. Max margin approaches, like SVMs, have been extremely successful in the classification community, offering state-of-the-art results [11]. In fact, one of the key findings of Zelenko *et al.* is that SVMs significantly outperform Voted Perceptron for relation extraction.

It can be shown that the key step in the SVM learning algorithm is the solution to:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$$

subject to $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$. This is a quadratic optimization problem, and there are multiple methods for solving this equation [24]. Again the kernel trick may be used since

the training data is only used during feature vector dot products. For both perceptron and SVMs, once α is found, it is possible to classify a new instance $\mathbf{x}$ by:

$$f(\mathbf{x}) = \begin{cases} \text{POS} & \text{if } \sum_i y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) \geq 0 \\ \text{NEG} & \text{otherwise.} \end{cases}$$

4.4 Discussion

There are two primary advantages of the approach taken by Zelenko *et al.* The first is that their system is able to exploit non-local dependencies since they are not required to model the parse structures of their system (unlike Miller *et al.*). This is explicitly handled through the similarity metric. Trees that share more substructure will be given a higher similarity score, making the function global in nature. The second advantage is that by reformulating the problem into a yes/no classification problem, they are able to take advantage of state-of-the-art discriminative methods like support vector machines.

To gain these advantages, there is a fundamental trade-off. Miller *et al.* propose a model in which decisions are made in unison, so that evidence for one decision may also provide evidence for another (i.e., phrase structure providing evidence for relations). Zelenko *et al.* must take the output of their parser as their gold standard and as a result is susceptible to pipeline errors. Having said that, noun-phrase chunks, part-of-speech and person/organization/location entity information are all well-studied problems with current systems displaying accuracy well above 90% [25, 26, 4]. Taking these parses as the gold standard may not result in too many pipelined errors.

It is not made clear by Zelenko *et al.* why they chose shallow parsing as the underlying structure for their kernel methods. Complete parse structure would probably provide more information, but automatic parsers might introduce more errors and the kernel computation will most likely be more cumbersome due to the added structure. Culotta and Sorenson [27] define a relation kernel over dependency structures, however they do not provide an empirical comparison to Zelenko *et al.*

As for the kernel function itself, it appears to be a good similarity metric. In fact, Zelenko *et al.* do present a very significant empirical improvement using their tree kernel as opposed to using a linear kernel within the SVM framework. However, in places the kernel seems a little restrictive. For instance, the indicator function $k(x, y)$ that is only on when substrings match exactly is excessive. A functions that takes into account string similarity, edit distance [28] or even word overlap might be more indicative. Clearly, it would be beneficial for the two strings *Bill Gates works at Microsoft* and *Microsoft employs Bill Gates* to contribute some weight to the similarity score. Similarly, by considering the parse structure for these two sentences, it can be seen that some score will be given for subsequences of length 1 (the matching noun phrases *Bill Gates* and *Microsoft*), however no weight will be given for a sequence of length 2 since the nodes appear in the wrong order, and the algorithm only considers ordered sequences. However, including unordered subsequences into the kernel calculation would result in a massive computational challenge.

This leads to the last point. As stated, the kernel computation in the worst cast is $O(k^2mn^3)$. To calculate the Gram matrix would then take $O(T^2k^2mn^3)$, where T is the size of the training data. Even with reasonable values of k, n, and T, this computation is

enormous. Zelenko *et al.* acknowledge this fact but provide no details as to the length of training and testing using their method.

5 Discussion and Future Directions

In each section, I presented multiple ways to improve upon the methods described and, when possible, gave a qualitative comparison between them. In this section a brief summary of these points is provided.

5.1 Quantitative Comparison?

Throughout this critical review I have failed to mention the empirical performance of each system on their varying tasks. I did so intentionally, since each system was evaluated with different data and relations of interest. Comparing numbers would be, at best, misleading. Instead I decided to focus on the qualitative aspects of each system. It would be an interesting experiment to test each system on the same set of data to check empirical performance. There exist so-called 'bake-off' competitions that serve just this purpose, such as MUC and its new incarnation ACE.

5.2 Semi-supervised Methods

One simple way to possibly improve semi-supervised approaches is to use trainable classifiers such as naive Bayes classifiers, decisions lists, perceptrons and support vector machines. An advantage of using such classifiers is that most provide a natural way of defining confidence of predictions through probability or scores. It may be tricky to incorporate some of these classifiers since most are quite powerful and could over-fit to the small amount of initial seed data. Some care would have to be taken, either through proper regularization or feature selection, in order to ensure that new examples are introduced. The classifiers also provide natural ways to handle non-local features such as those pertaining to document class and structure. For instance, it might be possible to train a separate classifier to recognize documents containing author information (e.g., Amazon.com pages). The output from this classifier could then provide beneficial features to a classifier that identifies author-book pairs in order to increase coverage.

There are also the considerations raised by Abney [13]. Abney only requires minor modifications to the original Yarowsky algorithm plus the use of EM or decision lists to show that the algorithm increases the likelihood of the data (or a related function). This would allow the semi-supervised methods to have stronger theoretical guarantees, which is a major advantage of supervised methods.

5.3 Supervised Methods

Unlike entity, part-of-speech and noun-phrase identification, which have been successfully extracted using local information only, relation extraction could benefit greatly from the use of non-local structural information. Generative parse models create tree structure, however,

state-of-the-art models do so by considering local information only. Recent advances in parse re-ranking [23] allow for the output of generative parses to be re-ranked with discriminative methods. The methods can incorporate more global information due to the limited search space for each training instance (usually < 30 trees).

Zelenko *et al.* avoid this problem entirely by assuming the output of a shallow parser is gold, and then defining tree kernels over the shallow parses. However, as noted, this reverts back to a pipelined method and is susceptible to errors common to such systems. A combination of the Miller *et al.* integrated parse method combined with a re-ranking method using Zelenko *et al.* kernel definitions would definitely be an experiment worth attempting.

There is also the question of appropriate data structure for representing relations. Miller *et al.* use full parse trees, whereas Zelenko *et al.* use only shallow parses. Recently Gildea and Palmer [29] showed that a system that uses full parses to extract semantic role information significantly outperformed one using only shallow parse information. This is very significant, since semantic role, or predicate-argument, identification is very similar to relation extraction. Both depend on identifying the presence of a relation/predicate as well as identifying the proper arguments.

It seems obvious that some complex syntactic structure should be beneficial, but there are actually many to choose from. For instance, Culotta and Sorenson [27] employ an approach similar to Zelenko *et al.*, but they use dependency trees as the underlying data structure. Dependency structure seems like a reasonable alternative since they naturally model verbs and their arguments, which is how many relations can be seen. Link grammars [30] are related to dependency grammars but also include syntactic information on connectives and could offer the advantages of parse-tree-like structural information combined with the improved parsing complexity of dependency grammars.

5.4 Semi-supervised vs. Supervised

It is hard to add more to the ongoing debate between semi-supervised and supervised approaches to not only relation extraction, but also natural language processing in general. From the literature, it seems that the community is definitely leaning towards supervised approaches when annotated data is available. Empirically, various bake-offs have also shown that supervised approaches outperform semi-supervised or unsupervised approaches on shallow parsing and entity extraction [4, 31]. Having said that, there are cases when little to no training data is available. Since annotated data can be expensive to create, semi-supervised approaches could prove highly useful in such situations.

5.5 Non-sentential Relations

A major deficiency of all systems studied here is that they focus primarily on sentential relations, in particular Miller *et al.* [7] and Zelenko *et al.* [9]. Relations are complex structures spanning multiple sentences and in some cases multiple documents. It would not be straightforward to modify the methods presented here to capture such long range relations. In particular, those methods relying on parse information would face inherent difficulty capturing these relations due to the sentential nature of parse trees.

References

[1] Berners-Lee T, Hendler J, Lassila O: **The Semantic Web**. *Scientific American* 2001, **284**(5):34–43.

[2] **World Wide Web Consortium**[[http://www.w3c.org]].

[3] Bikel DM, Schwartz R, Weischedel RM: **An algorithm that learns what's in a name**. *Machine Learning Journal Special Issue on Natural Language Learning* 1999, **34**(1/3):221–231.

[4] Tjong Kim Sang EF, De Meulder F: **Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition**. In *Proceedings of CoNLL-2003* 2003, pp.142–147.

[5] Agichtein E, Gravano L: **Snowball: extracting relations from large plain-text collections**. In *Proceedings of the fifth ACM conference on Digital libraries* 2000, pp.85–94.

[6] Yarowsky D: **Unsupervised Word Sense Disambiguation Rivaling Supervised Methods**. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics* 1995, pp.189–196.

[7] Miller S, Fox H, Ramshaw LA, Weischedel RM: **A Novel Use of Statistical Parsing to Extract Information from Text**. In *Proceedings of 1st Meeting of the North American Chapter of the Association for Computational Linguistics* 2000, pp.226–233.

[8] Collins M: **Three Generative, Lexicalised Models for Statistical Parsing**. In *Proceedings of the 35th Annual Meeting of the ACL* 1997.

[9] Zelenko D, Aone C, Richardella A: **Kernel Methods for Relation Extraction**. *Journal of Machine Learning Research* 2003, **3**:1083–1106.

[10] Collins M: **Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms**. In *Proceedings of Empirical Methods in Natural Language Processing* 2002.

[11] Joachims T: *Learning to Classify Text using Support Vector Machines.* Kluwer 2002.

[12] Blum A, Mitchell T: **Combining Labeled and Unlabeled Data with Co-training**. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers* 1998.

[13] Abney S: **Understanding the Yarowsky Algorithm**. *Computational Linguistics* 2004, **30**(3).

[14] Brin S: **Extracting Patterns and Relations from the World Wide Web**. In *WebDB Workshop at EDBT* 1998.

[15] Berger AL, Della Pietra SA, Della Pietra VJ: **A maximum entropy approach to natural language processing**. *Computational Linguistics* 1996, **22**.

[16] Steedman M, Osborne M, Sarkar A, Clark S, Hwa R, Hockenmaier J, Ruhlen P, Baker S, Crim J: **Bootstrapping Statistical Parsers from Small Datasets**. In *Proceedings of the Annual Meeting of the European Chapter of the ACL* 2003.

[17] Yangarber R: **Counter-Training in Discovery of Semantic Patters**. In *Proceedings of the 41st Annual Meeting of the ACL* 2003.

[18] Collins M: **Head-Driven Statistical Models for Natural Language Parsing**. *PhD thesis*, University of Pennsylvania 1999.

[19] Marcus M, Santorini B, Marcinkiewicz M: **Building a Large Annotated Corpus of English: the Penn Treebank**. *Computational Linguistics* 1993, **19**(2):313–330.

[20] Charniak E: **A Maximum-Entropy-Inspired Parser**. In *Proceedings of the North American Chapter of the Association for Computational Linguistics* 2000.

[21] Bikel D: **Intricacies of Collins Parsing Model**. *Computational Linguistics (to appear)* 2004.

[22] Goodman J: **Global thresholding and multiple-pass parsing**. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing* 1997.

[23] Collins M, Duffy N: **New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron**. In *In proceedings of the Annual Meeting of the ACL* 2002.

[24] Müller KR, Mika S, Rätsch G, Tsuda K, Schölkopf B: **An introduction to kernel-based learning algorithms**. *IEEE Neural Networks* 2001, **12**(2):181–201.

[25] Ratnaparkhi A: **A maximum entropy model for part-of-speech tagging**. In *IEEE Neural Networks* 1996.

[26] Sha F, Pereira F: **Shallow parsing with conditional random fields**. In *Proceedings of HLT-NAACL* 2003, pp.213–220.

[27] Cullota A, Sorensen J: **Dependency tree kernels for relation extraction**. In *Proceedings of the Annual Meeting of the Association for Computational Linguists* 2004.

[28] Cohen W, Ravikumar P, Feinberg S: **Comparison of String Distance Metrics for Name-Matching Tasks**. In *Proceedings of IIWeb workshop* 2003.

[29] Gildea D, Palmer M: **The Necessity of Syntactic Parsing for Predicate Argument Recognition**. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*.

[30] Sleator D, Temperley D: **Parsing English with a Link Grammar**. In *Proceedings of the Third International Workshop on Parsing Technologies* 1993.

[31] **A critical assessment of text mining methods in molecular biology workshop** 2004, [[http://www.pdg.cnb.uam.es/BioLINK/workshop/workshop_BioCreative_04]].