Syntactic Parsing for Ranking-Based Coreference Resolution

Altaf Rahman and Vincent Ng
Human Language Technology Research Institute
University of Texas at Dallas
Richardson, TX 75083-0688
{altaf, vince}@hlt.utdallas.edu

Abstract

Recent research efforts have led to the development of a state-of-the-art supervised coreference model, the cluster-ranking model. However, it is not clear whether the features that have been shown to be useful when employed in traditional coreference models will fare similarly when used in combination with this new model. Rather than merely re-evaluate them using the cluster-ranking model, we examine two interesting types of features derived from syntactic parses, tree-based features and path-based features, and discuss the challenges involved in employing them in the cluster-ranking model. Results on a set of Switchboard dialogues show their effectiveness in improving the cluster-ranking model: using them to augment a baseline coreference feature set yields a 8.6–11.7% reduction in relative error.

1 Introduction

Coreference resolution is the task of determining which noun phrases (NPs) in a text or dialogue refer to the same real-world entity. According to Webber (1979), coreference resolution can be decomposed into two complementary subtasks: “(1) identifying what a text potentially makes available for anaphoric reference and (2) constraining the candidate set of a given anaphoric expression down to one possible choice”. These two subtasks are commonly known as anaphoricity determination and anaphora resolution, both of which have recently been tackled using machine learning techniques. More specifically, anaphoricity determination is typically tackled by training an anaphoricity classifier, which determines whether an NP is anaphoric or not (e.g., Poesio et al. (2004), Zhou and Kong (2009)). If so, the NP is passed to the second component, the resolution system, which identifies an antecedent for the NP. This resolver is typically implemented by training a mention-pair (MP) model, which is a binary classifier that determines whether a pair of NPs are co-refering or not (e.g., Soon et al. (2001), Ng and Cardie (2002b)).

While this architecture is popularly adopted by coreference researchers and was implemented even within recently developed coreference resolvers (e.g., Bengtson and Roth (2008), Stoyanov et al. (2009)), neither the architecture itself nor its aforementioned implementation is satisfactory for at least two reasons. First, in this pipeline architecture, anaphoricity determination is performed prior to coreference resolution, so errors in anaphoricity determination can propagate to the downstream coreference component and adversely affect its performance (Ng and Cardie, 2002a). Second, the MP coreference model is fundamentally weak in that (1) the information extracted from two NPs may not be sufficient for making an informed coreference decision and (2) since the model is trained to compare the NP to be resolved (henceforth the active NP) against a candidate antecedent, it only determines how good the candidate is relative to the active NP, not how good the candidate is relative to other candidates.

In light of the aforementioned problems, researchers have proposed a number of solutions:

- To address the error propagation problem, researchers have proposed joint inference (Denis and Baldridge, 2007) and joint learning (Rahman and Ng, 2009) for anaphoricity determination and coreference resolution.
- To address the expressiveness problem resulting from making a coreference decision based on only two NPs, researchers have proposed the entity-mention model, where coreference decisions are made by determining whether an NP belongs to a preceding coreference cluster (e.g., Luo et al. (2004), Yang
et al. (2008)).

- To address the failure to directly compare candidate antecedents and determine the best one, researchers have proposed the mention-ranking model, which imposes a ranking on the candidate antecedents and therefore captures the competition among them (e.g., Dennis and Baldridge (2008), Iida et al. (2009)).

Recent research efforts have led to the development of a state-of-the-art supervised coreference model that can address all of the aforementioned problems, namely the joint cluster-ranking (CR) model (Rahman and Ng, 2009). However, other than its superior empirical performance to competing coreference models (such as the MP model), little is known about the joint CR model. In particular, most of the linguistic features for coreference resolution were developed and evaluated in the context of the MP model, and thus it is not clear whether these features would fare similarly when used in combination with the joint CR model.

Motivated by this observation, our goal in this paper is to examine the value of features derived from syntactic parses for the joint CR model. Note that parse-based features have been investigated extensively for the MP model. For example, they have been used to implement Binding Constraints (e.g., Luo and Zitouni (2005)) and encode syntactic salience (e.g., Haghighi and Klein (2009)). Rather than re-evaluate them for the CR model, we investigate two types of parse-based features that we believe are particularly interesting.

First, we employ parse trees directly as structured features for the joint CR model. The main advantage of employing tree-based structured features is simplicity: we no longer need to design heuristics to extract the desired features (e.g., salience, Binding Constraints) from the parse trees, as designing heuristics can be time-consuming and sometimes difficult for certain tasks. Note, however, that previous attempts have employed structured features to train an MP model for anaphora resolution (Yang et al., 2006; Versley et al., 2008) and an anaphoricity classifier in the aforementioned pipeline architecture (Zhou and Kong, 2009). In both cases, the structured features are combined with their non-structured (i.e., flat) counterparts via a composite kernel and used to train a classification model. What is interesting for us to investigate in this paper, however, is the question of how to combine flat and structured features in a ranking model that employs joint learning. With the increasingly important role structured features and ranking models play in natural language learning, we believe that a method for combining flat and structured features for training a ranker would be of particular interest to natural language processing (NLP) researchers.\footnote{The dual form of Collins and Duffy’s (2002) ranking algorithm can also combine flat and structured features. Note that their algorithm employs online learning, whereas ours employs batch learning in a maximum-margin fashion.}

Second, motivated in part by lexical semantics research (Lin and Pantel, 2001), we investigate path-based features, which encode the contextual relationship between an active NP and a candidate antecedent as the shortest path between the corresponding nodes in the parse tree. As with other NLP tasks, the effectiveness of a given type of features for coreference resolution depends in part on how the linguistic information it intends to capture is represented. We seek to investigate the extent to which a joint CR model can benefit from this path-based representation of context.

Unlike the vast majority of English coreference resolvers, which were evaluated using the MUC and ACE corpora, our resolver was evaluated on a set of Switchboard dialogues. To our knowledge, we are among the first to report results for the full coreference task on this dataset. As a result, our work contributes to the establishment of a baseline using a state-of-the-art supervised coreference model against which future work can be compared. Our experimental results indicate that while both the tree-based and path-based features improve coreference performance when applied to a Baseline feature set in isolation, the best performance is achieved when they are applied in combination. In particular, these two types of features yield an improvement of 2.2–3.7% in F-measure over the Baseline joint CR model, which corresponds to a 8.6–11.7% reduction in relative error.

The rest of the paper is organized as follows. Section 2 discusses our implementation of the joint CR model. Section 3 describes tree-based and path-based features and how they can be integrated into the CR model. We present evaluation results in Section 4 and conclude in Section 5.

# 2 The Baseline Coreference Model

This section describes the Baseline CR model. Since the CR model is a natural extension of the
2.1 The Mention-Pair Model

As noted before, the MP model is a classifier that determines whether two NPs are co-refering or not. Each instance \(i(NP_j, NP_k)\) corresponds to two NPs, \(NP_j\) and \(NP_k\), and is represented by 39 features (see Table 1 of Rahman and Ng (2009) for a description of these features). Linguistically, these features can be divided into four groups: string-matching, grammatical, semantic, and positional. However, they can also be categorized based on whether they are relational or non-relational: relational features capture the relationship between \(NP_j\) and \(NP_k\), whereas non-relational features capture the linguistic properties of one of them.

We follow Soon et al.’s (2001) method for creating training instances. Specifically, we create (1) a positive instance for each anaphoric NP \(NP_k\) and its closest antecedent \(NP_j\); and (2) a negative instance for \(NP_k\) paired with each of the intervening NPs, \(NP_j, NP_{j+1}, NP_{j+2}, \ldots, NP_{k-1}\). The classification associated with a training instance is either positive or negative, depending on whether the two NPs are coreferent. To train the MP model, we use the SVM learner from SVM\(^\textit{light}\) (Joachims, 1999).\(^2\)

After training, the classifier is used to identify an antecedent for an NP in a test text. Each NP, \(NP_k\), is compared in turn to each preceding NP, \(NP_j\), from right to left, and \(NP_j\) is selected as its antecedent if the pair is classified as coreferent. The process ends as soon as an antecedent is found for \(NP_k\) or the beginning of the text is reached.

2.2 The Cluster-Ranking Model

The CR model addresses two weaknesses of the MP model, one concerning expressiveness and the other concerning its failure to compare candidate antecedents directly and capture the competition among them. It does so by combining the strengths of the entity-mention model and the mention-ranking model. As discussed before, the mention-ranking model addresses the failure to compare candidate antecedents by training a ranker to impose a ranking on the candidate antecedents for an active NP. On the other hand, the entity-mention model addresses the expressiveness problem by determining whether an active NP belongs to a preceding, possibly partially-formed, coreference cluster. Its increased expressiveness stems from its ability to employ cluster-level features (i.e., features that are defined over any subset of NPs in a preceding cluster). Combining the entity-mention model and the mention-ranking model yields the CR model, which ranks the preceding clusters for an active NP so that the highest-ranked preceding cluster is the one to which the active NP should be linked.

Since the CR model ranks preceding clusters, a training instance \(i(c_j, \text{NONE})\) represents a preceding cluster \(c_j\) and an anaphoric NP \(\text{NONE}\). Each instance consists of two types of features: (1) features that are computed based solely on \(\text{NONE}\), and (2) cluster-level features, which describe the relationship between \(c_j\) and \(\text{NONE}\). Motivated in part by Culotta et al. (2007), we create cluster-level features from the relational features in our 39-feature set using four logical predicates: NONE, MOST-FALSE, MOST-TRUE, and ALL. Specifically, for each relational feature \(X\), we first convert \(X\) into an equivalent set of binary-valued features if it is multi-valued. Then, for each resulting binary-valued feature \(X_b\), we create four binary-valued cluster-level features: (1) NONE-X\(_b\) is true when \(X_b\) is false between \(\text{NONE}\) and each NP in \(c_j\); (2) MOST-FALSE-X\(_b\) is true when \(X_b\) is true between \(\text{NONE}\) and less than half (but at least one) of the NPs in \(c_j\); (3) MOST-TRUE-X\(_b\) is true when \(X_b\) is true between \(\text{NONE}\) and at least half (but not all) of the NPs in \(c_j\); and (4) ALL-X\(_b\) is true when \(X_b\) is true between \(\text{NONE}\) and each NP in \(c_j\).

We follow Rahman and Ng’s (2009) method for creating training instances. Specifically, for each NP, \(\text{NONE}\), we create a training instance between \(\text{NONE}\) and each preceding cluster \(c_j\) using the features described above. Since we are training a model for jointly learning anaphoricity determination and coreference resolution, we need to provide the ranker with the option to start a new cluster by creating an additional training instance that contains features that solely describe \(\text{NONE}\). The rank value of a training instance \(i(c_j, \text{NONE})\) created for \(\text{NONE}\) is the rank of \(c_j\) among the competing clusters. If \(\text{NONE}\) is anaphoric, the rank of \(i(c_j, \text{NONE})\) is HIGH if \(\text{NONE}\) belongs to \(c_j\), and LOW otherwise. However, if \(\text{NONE}\) is non-anaphoric, the rank of \(i(c_j, \text{NONE})\) is LOW unless \(c_j\) corresponds to the NULL cluster, in which case its rank is HIGH. Given these training instances, we can train a ranker using SVM\(^\textit{light}'s
After training, the cluster ranker processes the NPs in a test text in a left-to-right manner. For each active NP, $NP_k$, we create test instances for it by pairing it with each of its preceding clusters. To allow for the possibility that $NP_k$ is non-anaphoric, we create an additional test instance containing features that solely describe the active NP (as during training). All these test instances are then presented to the ranker. If the additional test instance is linked to the cluster that has the highest rank.

### 3 Tree-Based and Path-Based Features

In this section, we describe the tree-based and path-based features in detail and show how they can be exploited by the joint CR model.

#### 3.1 Path-Based Features

As mentioned before, a path-based feature encodes the contextual relationship between an active NP and a candidate antecedent as the shortest path between the corresponding nodes in the parse tree. More formally, a path between an active NP, $NP_k$, and a candidate antecedent, $NP_j$, in a parse tree is defined as the shortest sequence of nodes in the tree that need to be traversed in order to reach $NP_j$ from $NP_k$, and is represented as a sequence of non-terminal symbols, $s_1s_2...s_m$, where $s_i$ ($1 \leq i \leq m$) is the non-terminal symbol associated with the $i$th node being traversed in the path, with $s_1$ and $s_m$ being the non-terminal symbol associated with the nodes spanning $NP_k$ and $NP_j$, respectively. Given this representation, a path captures the shallow syntactic context in which two NPs appear.

There is a caveat, however. If the active NP and a candidate antecedent appear in different sentences, there will be no path between them. To enable the application of path-based features to these NPs, we create an additional “root” node with a random label (e.g., R) that connects the root nodes of the two trees containing these NPs. This allows a path to be established even if the two NPs appear in different sentences.

Now, to employ these paths for coreference resolution, two questions need to be answered. First, which paths should be used? In our implementation, we collect from each training text a path between each NP and each of its preceding NPs.

This yields approximately 512K paths. For efficiency reasons, we reduce the number of paths being considered by removing those paths that occur less than seven times in the training set. After this filtering process, only approximately 22K paths remain. Each resulting path is represented as a binary-valued feature for coreference resolution. Second, how can we compute the value of a path-based feature? If we were to train an MP model, its value is 1 if the path between the two NPs under consideration is the same as the path represented by the feature. Otherwise, its value is 0. Since we are training a joint CR model, where each instance corresponds to an NP, $NP_k$, and a preceding cluster, $c_j$, rather than two NPs, we compute its feature value as follows: its value is 1 if the path between $NP_k$ and one of the NPs in $c_j$ is the same as the path represented by the feature; otherwise, its value is 0.

We hypothesize that by capturing shallow syntactic context, path-based features can improve the performance of a coreference system. The reason is that through these features, a learner can potentially learn to distinguish between good paths (i.e., paths that are likely to connect coreferent NPs) and bad paths (i.e., paths that are likely to connect non-coreferent NPs), thus improving the resulting model’s ability to identify the correct antecedent or preceding cluster for an active NP.

#### 3.2 Tree-Based Features

Not only can parse trees be exploited to identify coreference relations via the extraction of paths, but they can be used to determine the anaphoricity of an NP. Specifically, we aim to identify non-anaphoric NPs by employing parse trees as structured features. While previous work has employed parse trees as structured features (Zhou and Kong, 2009), it does so in a pipeline architecture where anaphoricity determination is performed prior to coreference resolution. In contrast, we are faced with the challenge of integrating tree-based structured features with flat features in a model that involves both joint learning and ranking.

To understand how this can be done, recall that in the joint CR model, joint learning for anaphoricity determination and coreference resolution is achieved by introducing an additional training instance, $i(NULL, NP_k)$, which is formed between an active NP, $NP_k$, and a NULL preceding cluster, effectively providing $NP_k$ with an option to start a
new cluster. Since we aim to use tree-based features to identify non-anaphoric NPs, we augment the set of features for \(i(\text{NULL, } \text{NP}_k)\), which currently contains the flat features derived from \(\text{NP}_k\), with these (structured) tree-based features.

Of course, having an SVM learner learn a ranking model from both the flat and tree-based features requires more than just adding the tree-based features to the feature set. In particular, we need to implement the three steps below.

**Step 1: Specifying the Parse Substructure**

While we want to use a parse tree directly as a feature, we do not want to use the entire tree as a feature. The reason is that a complex tree may make it difficult for the SVM learner to make generalizations: the more complex the tree is, the less likely it is to find similar trees in other instances.

To strike a better balance between having a rich representation of context and improving the learner’s ability to generalize, we extract a substructure from a parse tree and use it as the value of the structured feature of an instance. This substructure was previously shown to be useful when used as a structured feature for training a classifier for determining the information status of an NP (Rahman and Ng, 2011). Given an instance \(i(\text{NULL, } \text{NP}_k)\), we extract the substructure from the parse tree containing \(\text{NP}_k\) as follows. Let \(n(\text{NP}_k)\) be the root of the subtree that spans all and only the words in \(\text{NP}_k\), and let Parent\((n(\text{NP}_k))\) be its immediate parent node. We (1) take the subtree rooted at Parent\((n(\text{NP}_k))\), (2) replace each leaf node in this subtree with a node labeled \(X\), (3) replace the child nodes of \(n(\text{NP}_k)\) with a leaf node labeled \(Y\), and (4) use the subtree rooted at Parent\((n(\text{NP}_k))\) as the structured feature for \(i(\text{NULL, } \text{NP}_k)\). Figure 1 illustrates this substructure extraction procedure via an example.

Intuitively, the first three steps aim to provide generalizations by simplifying the tree. For example, step (1) allows us to focus on using a small window surrounding \(\text{NP}_k\) as its context. Steps (2) and (3) help generalization by ignoring the words within \(\text{NP}_k\) and its context. Note that using two labels, \(X\) and \(Y\), helps distinguish the active NP from its context within this substructure. Also note that we simply use one node (\(Y\)) to represent the active NP; since NP-internal information (e.g., gender) has been captured by the flat features.

While this parse substructure ignores the words in \(\text{NP}_k\), these unigrams could be useful for determining its anaphoricity, as a learner may learn from coreference-annotated data that “it” only has a moderate probability of being anaphoric, and that “the contrary” from the phrase “on the contrary” is never anaphoric. As a result, we augment the set of flat features in \(i(\text{NULL, } \text{NP}_k)\) with the unigrams extracted from \(\text{NP}_k\).

**Step 2: Recasting Ranking as Classification**

Existing implementations of SVMs, such as SVMlight-TK (Moschitti, 2004), allow us to combine flat and (structured) tree-based features to train a classifier by designing appropriate kernels. Hence, if we were to train an SVM classifier, all we need to do is to design a kernel. However, we are given a ranking problem, and it is not immediately clear how an SVM can learn a ranking model in the presence of tree-based features.

Our approach to this problem is to reduce the given ranking problem to an equivalent classification problem. Once we have a classification problem, all we need to do is to design a kernel for training a classifier, as mentioned above. To reduce a ranking problem to an equivalent classification problem, we need to convert the training set for the joint CR model to an equivalent training set that can be used to train a classifier.

Before describing the conversion process, let us first recall how the training set for a joint CR model is created. Given a training text \(D\), we create from \(D\) a set of training instances \(T\) for a joint CR model by taking the union of \(T_1, T_2, \ldots, T_n\), where \(T_k\) (\(1 \leq k \leq n\)) is the set of training instances generated from \(\text{NP}_k\) in \(D\). If \(\text{NP}_k\) has \(|C|\) preceding clusters, \(T_k\) will contain exactly \(|C| + 1\) training instances, since one training instance is generated from \(\text{NP}_k\) and each of its \(|C|\) preceding clusters, and one training instance is formed between \(\text{NP}_k\) and the NULL antecedent. Each instance is associated with a rank value, which is either HIGH or LOW. Given \(T\), the SVM ranker-

![Figure 1: A parse tree (left) and the parse substructure extracted for the NP “one day” (right).](image-url)
learning algorithm aims to learn how to rank preceding clusters for an active NP by learning how to rank the instances within each $T_k$.

As noted before, to facilitate learning a ranker from both flat and tree-based features, we reformulate the given ranking problem as a set of pairwise ranking problems. The reason is that a pairwise ranking problem is essentially a binary classification problem, since pairwise ranking merely involves ranking two objects. Not surprisingly, this reformulation requires that we convert $T$ into an equivalent training set $T'$, which consists of pairwise ranking problems and can therefore be used to train a classifier (i.e., a pairwise ranker). Below we describe how to convert $T$ to $T'$.

For each $T_k$ in $T$, we create a training instance $inst$ for $T'$ from each pair of training instances in $T_k$ that have different rank values. For example, if $i(c_j, NB_k)$ and $i(c_j, NB_k)$ in $T_k$ have ranks $r_1$ and $r_2$ respectively where $r_1 \neq r_2$, we create a training instance for $T'$ whose feature vector is obtained by subtracting $i(c_j, NB_k)$ from $i(c_i, NB_k)$. If both feature vectors contain only flat features, the subtraction is straightforward, since each flat feature is real-valued. However, if one of the feature vectors has a tree-based feature3 (which happens when $c_i$ or $c_j$ is NULL), we handle the flat features and the tree-based feature separately. Specifically, we first perform subtraction for the flat features as described above, and then append the tree-based feature to the feature set of $inst$. If $r_1 > r_2$, the class value of $inst$ is 1; otherwise, it is −1.

In sum, each $T_k$ in $T$ constitutes a ranking problem, and we described how to convert this ranking problem into a set of pairwise ranking problems in $T'$. As noted before, a pairwise ranking problem is a binary classification problem. Hence, the resulting training set $T'$ can be used to train a (binary) SVM classifier that minimizes the number of violations of pairwise rankings in $T'$.

3Note that at most one of these two feature vectors has a tree-based feature. The reason is that exactly one of the instances in $T_k$ has a tree-based feature, namely the one corresponding to the NULL cluster.

Step 3: Designing the Composite Kernel
To train an SVM classifier on $T'$, we need to define a kernel function for computing the similarity between a pair of instances. If both instances contain only flat features, we simply employ a normalized linear kernel, which computes similarity as the cosine of their feature vectors. However, if one or both of them has a tree-based feature, a linear kernel is not directly applicable. In this case, we need to (1) compute the similarity of their flat features and the similarity of their tree-based features separately, and then (2) employ a composite kernel, $K_c$, to combine the two similarity values. Specifically, we define $K_c$ as follows:

$$K_c(F_1, F_2) = K_1(F_1, F_2) + \alpha K_2(F_1, F_2),$$

where $F_1$ and $F_2$ are the full set of features (containing both flat and structured features) that represent the two instances under consideration. $K_1$ is a linear kernel, which operates on the flat features. $K_2$ is a convolution tree kernel (Collins and Duffy, 2001), which operates on the tree-based features. Specifically, $K_2$ computes the similarity of two parse trees by efficiently enumerating the number of common substructures in them. To prevent the kernel value returned by $K_c$ from being consistently dominated by one of the component kernels (i.e., $K_1$ and $K_2$), we normalize the kernel values returned by $K_1$ and $K_2$ so that they fall between 0 and 1. $\alpha$ is as a weight parameter that allows the two kernel values to be combined linearly, providing the flexibility to vary the relative importance of the component kernels. We will determine $\alpha$ empirically on the development set.

3.3 Applying the Pairwise Ranker
So far, we have described a method for training a (pairwise) ranker when the feature set contains both flat and tree-based features, which involves converting training set $T$ to training set $T'$. A natural question, then, is: do we have to similarly perform this conversion on the test set so that the pairwise ranker can be applied to it?

It turns out that the answer is no. Given a set of test instances $T_k$ to be ranked, all we need to do is to apply the pairwise ranker to each instance in $T_k$. The ranker produces one real value for each instance. According to the values provided by the ranker, these test instances can be ranked: the most positive value corresponds to the highest rank.

It may not be immediately clear why it makes sense to apply the pairwise ranker in the aforementioned manner to rank the test instances. Space limitations preclude a rigorous mathematical explanation. Here, we will provide a sketch of the explanation. Recall that each instance in $T'$ was converted by subtracting the feature vectors of two instances. In addition, when SVMlight was applied to train the pairwise ranker on $T'$, it attempted to minimize the number of violations of...
pairwise rankings. To do so, SVM\textsuperscript{light} needs to position the hyperplane so that an instance with a higher rank in \( T \) is assigned a more positive value by the hyperplane than one with a lower rank in \( T \). Consequently, we can apply the pairwise ranker to each test instance to be ranked, and use the value returned by the ranker for each instance to impose a ranking on the test instances.

4 Evaluation

In this section, we examine the effectiveness of the tree-based and path-based features in improving the joint CR model.

4.1 Experimental Setup

\textbf{Corpus.} We employ in our evaluation a dataset comprising 147 coreference-annotated Switchboard dialogues, which contain a total of 68,992 NPs.\textsuperscript{4} We partition the dialogues into a training set (117 dialogues) and a test set (30 dialogues). We extract the NPs and the parse trees directly from the gold-standard annotations, but the coreference features are computed entirely automatically.

\textbf{Scoring programs.} We employ two commonly-used coreference scoring programs, B\textsuperscript{3} (Bagga and Baldwin, 1998) and \( \phi_3 \)-CEAF (Luo, 2005), both of which report results in terms of recall (R), precision (P), and F-measure (F).

4.2 Results and Discussion

\textbf{The baseline mention-pair model.} We employ as our first baseline the MP model, which is trained using the procedure described in Section 2.1. Given that our goal is to examine the effectiveness of the tree-based and path-based features for the joint CR model, one may wonder why the results of the MP model are relevant to our investigation. Recall from the introduction that we chose to improve the joint CR model with the two types of features derived from syntactic parses because the joint CR model has been shown to achieve state-of-the-art performance on the ACE corpus. To ensure that the joint CR model also outperforms the MP model on our Switchboard corpus (and is therefore the strongest baseline we can use), we show the results of the MP model in row 1 of Table 1. As we can see, it achieves F-measure scores of 69.1 (B\textsuperscript{3}) and 62.8 (CEAF).\textsuperscript{5}

\textbf{Incorporating path-based features.} Next, we incorporate the path-based features into the Baseline joint CR model. Results are shown in row 3 of Table 1. In comparison to the results of the Baseline joint CR model in row 2, we can see that adding the path-based features into the feature set improves the joint CR model according to both scorers. In particular, B\textsuperscript{3} and CEAF F-measure scores rise by 1.3% and 2.1%, respectively, suggesting the usefulness of the path-based features.

In addition to the R, P and F columns, Table 1 has two columns labeled “% err. red.”, which show the error reduction of a system relative to the Baseline joint CR model. Here, we compute the error of a system by subtracting its F-measure score from the perfect F-measure (i.e., 100). With the addition of path-based features, we can see that relative error is reduced by 5.1 and 6.7 according to B\textsuperscript{3} and CEAF, respectively.

\textbf{Incorporating tree-based features.} Next, we incorporate the tree-based features into the Baseline joint CR model. Recall that from a tree, we extract both flat features (i.e., unigrams) and structured features (i.e., parse substructures), so both types of features are used to augment the Baseline feature set. Because both types of features are involved, we need to tune \( \alpha \) in the composite kernel. To ensure a fair comparison among different systems, we do not employ additional labeled data for tuning \( \alpha \). Rather, we use 75% of the available training data for training the joint CR model and reserve the remaining 25% for parameter tuning.

Results are shown in row 4 of Table 1. In comparison to the results of the Baseline joint CR model in row 2, we can see that adding the trees and the unigrams into the feature set improves the joint CR model according to both scorers. In particular, B\textsuperscript{3} and CEAF F-measure scores rise by 1.0% and 1.9%, respectively.

\footnotesize{\textsuperscript{4}This dataset is released by the LDC as part of the NXT corpus (Calhoun et al., 2010).

\textsuperscript{5}Since gold-standard NPs are used in our coreference experiments, CEAF recall, precision, and F-measure will all be the same. See Luo (2005) for details.}
Table 1: Coreference results on the test set obtained using B3 and CEAF.

<table>
<thead>
<tr>
<th>System</th>
<th>B3 R</th>
<th>B3 P</th>
<th>B3 F</th>
<th>CEAF R</th>
<th>CEAF P</th>
<th>CEAF F</th>
<th>% err. red.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline MP model</td>
<td>78.1</td>
<td>61.6</td>
<td>69.1</td>
<td>—</td>
<td>62.8</td>
<td>62.8</td>
<td>—</td>
</tr>
<tr>
<td>Baseline CR model</td>
<td>71.1</td>
<td>78.2</td>
<td>74.5</td>
<td>—</td>
<td>68.5</td>
<td>68.5</td>
<td>—</td>
</tr>
<tr>
<td>CR + paths</td>
<td>76.4</td>
<td>75.2</td>
<td>75.8</td>
<td>(5.10)</td>
<td>70.6</td>
<td>70.6</td>
<td>(6.67)</td>
</tr>
<tr>
<td>CR + unigrams + trees</td>
<td>75.1</td>
<td>76.0</td>
<td>75.5</td>
<td>(3.92)</td>
<td>70.4</td>
<td>70.4</td>
<td>(6.03)</td>
</tr>
<tr>
<td>CR + paths + unigrams + trees</td>
<td>76.6</td>
<td>76.8</td>
<td>76.7</td>
<td>(8.65)</td>
<td>72.2</td>
<td>72.2</td>
<td>(11.74)</td>
</tr>
<tr>
<td>CR + paths + unigrams</td>
<td>76.3</td>
<td>75.4</td>
<td>75.8</td>
<td>(5.10)</td>
<td>71.5</td>
<td>71.5</td>
<td>(9.52)</td>
</tr>
<tr>
<td>CR + paths + pipeline architecture</td>
<td>76.9</td>
<td>75.2</td>
<td>76.0</td>
<td>(5.88)</td>
<td>71.4</td>
<td>71.4</td>
<td>(9.21)</td>
</tr>
</tbody>
</table>

Incorporating tree- and path-based features. Next, we incorporate both tree-based (i.e., unigrams and parse substructures) and path-based features into the Baseline joint CR model. As in the previous experiment, we reserve 25% of the available training data for tuning $\alpha$. Results are shown in row 5 of Table 1. In comparison to the results of the Baseline joint CR model in row 2, we can see that adding both types of features improves F-measure by 2.2% (B3) and 3.7% (CEAF), which is equivalent to a relative error reduction of 8.6% (B3) and 11.7% (CEAF).

In comparison to the results in rows 3 and 4, we can see that better results can be obtained by applying the two types of features in combination than in isolation to the Baseline joint CR model. This suggests that although both types of features are derived from parse trees, they provide complementary information for the CR model.

Understanding the value of parse substructures. So far, we have always applied the unigrams and the parse substructures in combination in our experiments. To better understand the value of the parse substructures, we perform an ablation experiment in which we repeat the previous experiment without using the parse substructures.

Results are shown in row 6 of Table 1. In comparison to the results in row 5, we can see that F-measure drops by 0.9% (B3) and 0.7% (CEAF). Since the difference in results between the two rows can be attributed entirely to the presence/absence of the parse substructures, the drop in F-measure suggests that the parse substructures are indeed useful features for the joint CR model.

Pipeline vs. joint modeling. One challenge we addressed here involves enabling the integration of structured and flat features in a ranker that performs joint learning. A natural question is: is this joint learning architecture indeed better than the traditional pipeline architecture in which anaphoricity determination is performed prior to coreference resolution? To answer this question, we show in row 7 of Table 1 the results obtained using the pipeline architecture, where (1) an anaphoricity classifier is trained with all the features used to represent an instance involving the NULL antecedent in the joint CR model in row 5 and (2) the joint CR model is trained using the Baseline and path-based features. This setup would therefore allow us to determine whether the joint architecture or the pipeline architecture can better exploit the structured features. In comparison to the results in row 5, we see that F-measure drops by 0.6–0.8%. These results suggest that joint learning is indeed better than pipeline learning in terms of exploiting structured features.

5 Conclusions

We have examined the effectiveness of tree-based and path-based features in improving a state-of-the-art supervised coreference model, the cluster-ranking model. Results on 147 Switchboard dialogues, show that both types of features are effective at improving the performance of the cluster-ranking model. In particular, when they are applied in combination, we see a reduction in relative error by 8.6–11.7%. One challenge that we addressed during the course of this investigation involves enabling flat and structured features to be employed simultaneously in a ranking model that employs joint learning. With the increasingly important role structured features and ranking models play in natural language learning, we believe that our method for combining flat and structured features for training a ranker would appeal to researchers working in different areas of NLP.

Acknowledgments

We thank the three anonymous reviewers for their invaluable comments on an earlier draft of the paper. This work was supported in part by NSF Grants IIS-0812261 and IIS-1147644.
References


