Reinforcement Learning for Coreference Resolution

Aishwarya P (CS11B004), Dhivya E (CS11B012),
Varshaa Naganathan (CH11B070)

May 11, 2015

Abstract

Coreference resolution is an important step for a number of higher level NLP tasks that involve natural language understanding. Conventional methods treat this task as a classification problem [2] where a pair of mentions are classified as coreferents or as a clustering problem where each cluster represents a named entity. The availability of partial structured information for supervision motivates the formulation of this problem in a reinforcement learning setting. Using similarity between two clusterings as our reward, we propose to employ function approximation to learn generalizable rules for coreference resolution.

1 Introduction

Natural languages provide speakers with a variety of ways to refer to entities. Coreference resolution is the process of determining whether two such expressions refer to the same entity in the world. More specifically, two noun phrases are said to be co-referring to each other if both of them resolve to a unique referent unambiguously [2]. The referring expressions that participate in the coreference relation are called mentions. A coreference resolution system is expected to take a document as input and produce clusters of mentions that refer to the same entity. Most systems return chains of coreferring mentions, where the mentions are ordered as in the original text. Coreference resolution has important applications in higher-level Natural Language Processing tasks such as question answering, machine translation and automatic text summarization.

Traditional solution methods to coreference resolution typically follow a two-phase approach, first classifying pairs of mentions into whether or not they refer to the same entity and then accumulating this information across all pairs of mentions to obtain clusters of mentions referring to the same entity. However, since the first phase cannot make use of global information, it is likely to result in conflicts, for example, indicating that mentions A and B refer to the same entity, mentions B and C refer to the same entity but mentions A and C refer to different entities.

2 Motivation

The agglomerative approach to clustering involves a sequence of decisions according to which existing clusters are combined to create new clusters. However, it assumes that the metric used to make decisions is known a-priori and remains constant throughout the process, assumptions which may not hold in real applications like our problem of interest. Ideally, we would want to *learn* the possibly varying metric by a series of exploratory and exploitative decisions of coalescing clusters, a situation that maps naturally into the reinforcement learning setting.

One such reinforcement learning based model for coreference resolution was suggested by Stoyanov *et al.*[4] In this model, the current state consists of all current partially formed coreference clusters, the start state has each mention being considered as a separate single-element cluster and the available actions at a state include joining existing clusters or halting and returning the current clustering. To allow this formulation to generalize across documents, a feature representation for states and actions has been adopted. However, Stoyanov *et al.*[4] do not make use of rewards to update the model parameters, as is done in standard reinforcement learning problems, preferring to use perceptron style updates.

3 Our Framework

We wish to extend the model from Stoyanov *et al.*[4] to make use of rewards that indicate the goodness of the current clustering. This can be done as follows - for every training document in the corpus, the ground truth coreference clusters are available. We can compare the current clustering with the ground truth using measures such as the adjusted RandIndex [5]. The reward for performing an action is the difference in scores according to such a metric before and after performing the action. This framework now allows for the use of a wide range of function approximation based reinforcement learning algorithms to learn a policy for clustering.

3.1 States

As in Stoyanov *et al.*[4], our states hold information about which mentions are currently clustered together. The list of properties used to describe a mention can be found in appendix A.

Since clusters in the context of coreference resolution are usually small in size, we prefer to keep track of individual mentions in every cluster as opposed to summary statistics such as the centroid or medoid. In the start state, as in agglomerative clustering, each mention is in a separate cluster. Over time, as actions are performed, clusters of mentions get merged together.

3.2 Actions

At any state, the allowed actions are the merging of two existing clusters. Actions are represented using features obtained from the properties of the two mentions in the clusters under consideration. This feature representation is chosen to allow learning of policies that generalize across documents. Since the state space - all possible clusterings of all possible documents is very large, it is not possible to learn the exact state-action value for every action at every state.

To obtain the feature values corresponding to the merging of clusters C and C', we aggregate the feature values of pairs of mentions (m, m') where $m \in C$ and $m' \in C'$ as follows:

- For numeric features, the average value across all such pairs of mentions is used.
- For nominal features, the majority value across all such pairs of mentions is used.

The features for a pair of mentions are those used commonly in two-phase coreference systems [3]. For better comparison, we use the same features that our baseline system - Reconcile [3] uses. The list of features used can be found in appendix B.

3.3 Rewards

Our reward is based on the similarity between the current clustering and the true clustering, available for training documents. Consider any pair of mentions. If they belong to the same cluster in both clusterings, the pair is treated as a true positive. If they belong to the same cluster only in the true clustering, then the pair is false negative. If both mentions are in the same cluster in the current clustering but not in the true clustering, the pair is an instance of a false positive. If they are in different clusters in both the clustering, then it is a true negative. Using these, the following measures of similarity have been computed and used as reward for reaching the current state.

- **RandIndex:**

$$\text{RandIndex} = \frac{\text{True positives} + \text{True Negatives}}{\text{True positives} + \text{True Negatives} + \text{False positives} + \text{False Negatives}} \tag{1}$$

- **Recall:**

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False Negatives}} \tag{2}$$

- **Precision:**

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{3}$$

- **RandIndex Difference:**

  The difference between the randIndex of the current state and the previous state.

- **Recall Difference:**

  The difference between the recall of the current state and the previous state.

- **Precision Difference:**

  The difference between the precision of the current state and the previous state.

Intuitively, using difference of similarity measures of the current and previous clustering with the true clustering makes more sense because we ideally need the rewards to be indicative of the extent to which an action improves the clustering. In this manner an action that leads to the decrease in similarity value gets a negative reward, which we expect to be more effective than using the similarity measure directly as the reward.

3.4 Q-Function

3.4.1 Linear Function Approximation

Since we do not have an a-priori belief of the shape of the reward function in the feature space, a standard choice of function approximation for the Q-function was a linear function. In this method, each feature ϕ_i has a corresponding weight θ_i and the Q-value of action a in state s is obtained as

$$Q(s, a) = \sum_i \theta_i \phi_i(s, a) \tag{4}$$

The weights can now be learnt using standard reinforcement learning algorithms such as SARSA(λ) or Watkin's Q(λ).

3.4.2 Fitted Q-Iteration

Linear function approximation imposes a strong bias on the state-action value function. A better method would be to use a more general model such as regression trees or neural networks to learn the Q-values. Further, since most of our features are categorical, we can expect regression trees to perform better than techniques that assume all features are numerical (such as linear regression).

However, the learning of models such as regression trees requires a batch update in contrast to the online updates used in linear function approximation. For this, we use fitted Q-iteration. We generate trajectories according to the current policy by clustering a few training documents using the current model and collect reward samples for each step of the clustering. These samples are then used as training data to update the model learnt.

3.5 Stopping Condition

An important decision to be made in agglomerative hierarchical clustering is when to stop merging clusters. A few ways to incorporate this into our framework is to either stop when no action has a Q-value above a threshold or to stop when Q-values of two successive greedy actions differ by more than a threshold. However, the true number of clusters to be obtained is dependent on the document. It depends on factors such as the size of the document and the number of topics dealt with in it. Since our reward functions compare clusterings, they will be affected by the true number of clusters. Thus, our Q-values are also dependent on the document. This makes it difficult to set a common threshold to use as a stopping condition for all documents.

Empirically also, we found that such Q-threshold based methods were not effective in stopping the clustering at a good point. This is possibly because resolution between different types of entities require different thresholds. Hence, we used a different heuristic as the stopping condition - we stop when the number of clusters falls to a certain fraction of the original number.

4 Evaluation

Since the main objective of the task is to perform coreference resolution, we use the following standard coreference resolution metrics to evaluate our system.

4.1 MUC-Recall

MUC-Recall is defined as follows. The true clustering partitions the set of all mentions into some subsets $S_i,\ i = 1, 2, \ldots n$. Also, the clustering generated by our system partitions the mentions into sets $R_j,\ j = 1, 2, \ldots m$. Then, we can partition each set S_i into subsets such that all mentions in a subset belong to the same cluster according to our clustering. These subsets will be $S_i \cup R_1, S_i \cap R_2, \ldots S_i \cap R_m$. Note that some of these could be empty. We define $p(S_i) = \{S_i \cap R_j : S_i \cap R_j \neq \phi\}$. Then, the MUC-Recall is calculated as

$$\text{MUC-Recall} = \frac{\sum_i (|S_i| - |p(S_i)|)}{\sum_i (|S_i| - 1)} \tag{5}$$

Intuitively, the MUC-Recall captures the extent to which true clusters are maintained in the system's clustering. A system will get high recall if it does not split any true clusters. Note that it is possible that two true clusters may get merged together. In fact, merging all mentions into a

single cluster results in very high recall. Due to this, another measure is needed in conjunction with recall, for which we turn to MUC-Precision.

4.2 MUC-Precision

MUC-Precision aims to determine how likely it is that mentions that have been clustered together by the system actually should be present in the same cluster. Following the same notation as in recall, we define $p(R_j) = \{S_i \cap R_j : S_i \cap R_j \neq \phi\}$. Then, the MUC-Precision is given by,

$$\text{MUC-Precision} = \frac{\sum_j (|R_j| - |p(R_j)|)}{\sum_j (|R_j| - 1)} \tag{6}$$

5 Experiments

5.1 Baseline

As a baseline for comparison, we use the system Reconcile developed by Ng and Cardie [3]. The Reconcile system was primarily developed to address the issue of poor recall in existing systems. The MUC scores of Reconcile are as follows -

MUC-Precision	0.4699928097
MUC-Recall	0.7104158644

5.2 Dataset

The dataset used was the University of Wolverhampton corpus supplied with Reconcile. This has 104 documents and we use 70 as a train set and 34 as a test set. Some statistics of this dataset are as follows -

Average number of clusters = 83.38461
Average number of mentions = 131.27884
Average number of mentions per cluster = 1.578835

5.3 Results and Discussion

Linear function approximation with Q-learning was found to diverge. Hence, we used SARSA(λ) to learn the weights. For linear function approximation with SARSA(λ), we explored the variation of performance (MUC-Precision and MUC-Recall) with a number of parameters of the system, such as the choice of reward function, the learning rate α, the discount factor γ, the greedy coefficient ϵ, λ which controls the rate of decay of eligibility traces and the stopping threshold.

The variation of performance with reward function for $\lambda = 1, \gamma = 0.2, \epsilon = 0.1$ and $\alpha = 0.1$ and cluster threshold 0.8 is given in Table 1.

Reward Function	MUC-Precision	MUC-Recall
Recall	0.6180894471	0.3648169411
Recall Difference	0.6180894471	0.36481694-11
RandIndex	0.6180894471	0.3648169411
RandIndex Difference	0.6180894471	0.3648169411
Precision	0.6180894471	0.3648169411
Precision Difference	0.06885274799	0.04002934825

Table 1: Variation of performance with reward function

It is surprising that the choice of reward function does not often affect the final clustering obtained for the test documents. We had expected using a difference in precision, recall or RandIndex to perform better than using the similarity directly but this was not found to be the case. In the only case where there is a difference, with precision, the performance is drastically lowered.

An important parameter that is seen to affect the performance is the stopping condition. We stop clustering when the number fo clusters falls to a certain fraction of the original number. The variation of performance with this fraction can be seen in Figure 1.

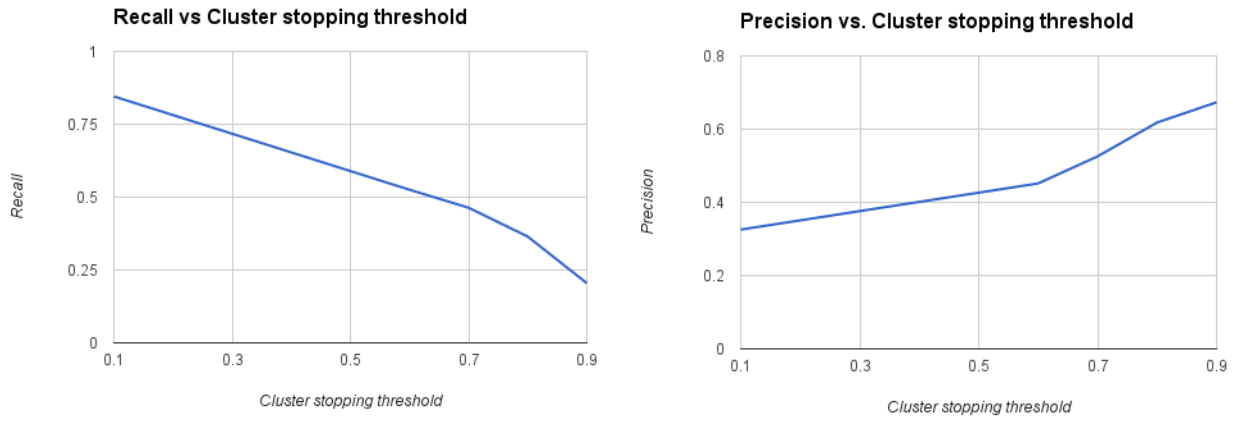

Figure 1: Variation of performance with stopping condition

Initially, when the threshold is low, a significant amount of merging happens. So recall is high and precision is low. As the threshold is increased, the number of clusters in the final clustering increases and we see an improvement in precision, which is traded-off with a drop in recall. For most values of the threshold, we perform better than Reconcile in terms of precision but require very small values of the threshold to outperform them in terms of recall. Unfortunately, we do not have a single value of the threshold for which we outperform Reconcile in terms of both precision and recall.

We attempted tuning parameters such as γ, α, ϵ and λ. We found that tuning γ, α or ϵ did not change the final clustering. While this is expected for ϵ and α, as they only affect the learning process, not change the underlying MDP, it is surprising that changing γ did not change the solution obtained.

Changing λ however, affects the clustering learnt. For some values of λ, the performance is found to be very poor. The trend can be seen in Figure 2.

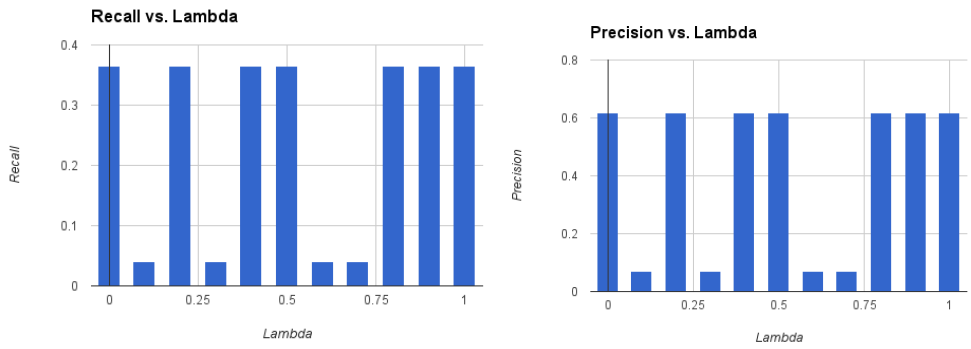

Figure 2: Variation of performance with λ

Instead of linear function approximation, regression methods can be used to estimate a value function with less bias if we can train in batch mode. This is done using fitted Q-iteration. We attempted to model the value function first using a regression tree and a multilayer perceptron. When training using fitted Q-iteration, we first obtain a few samples using the current model, train a new model, then obtain samples again using the new model. When training a new model an important decision is the weight given to samples taken using earlier models. Ideally we would like to use only the samples from the most recent model, but this could result in a scarcity of data to train the model. Hence we decay the weights given to the earlier samples. A decay factor of 1 would mean that old samples have as much weight as recent samples whereas a decay factor of 0 would mean that old samples are never re-used. Figure 3 shows the variation of performance with a regression tree with the weight decay factor. The values of γ and ϵ have been fixed at 0.2 and 0.1 respectively for this experiment.

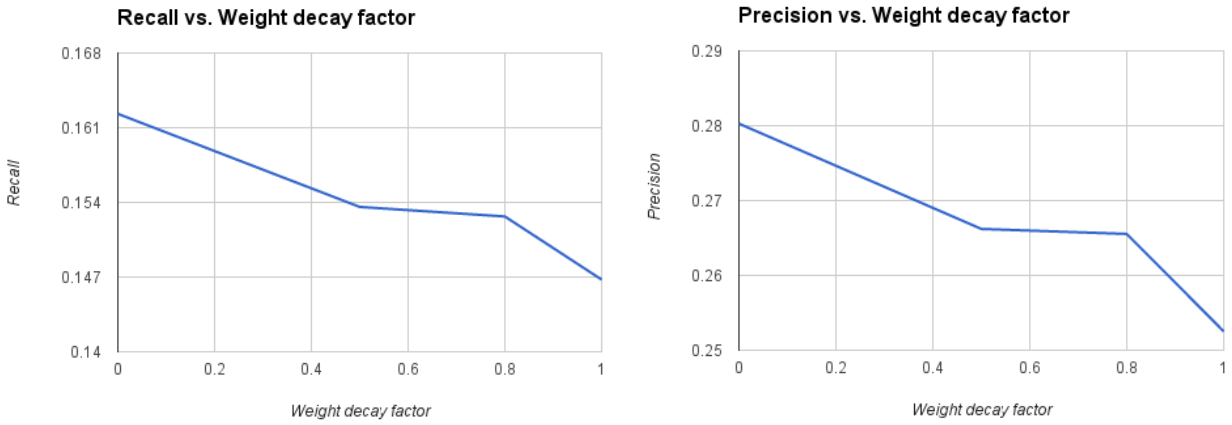

Figure 3: Variation of performance with weight decay factor

We observe that the performance is very poor in comparison to the linear function approximation. This is possibly because the training set is not large enough for the feature set. We also attempted to use a multilayer perceptron instead of a regression tree to learn Q-values. Intuitively, since many of the features being used are nominal features, the performance is not expected to be as good. We also attempted varying the number of documents whose clustering is included in a batch. Note that a single document provides about 100 training samples. The variation of the performance of the multilayer perceptron with the number of documents in a batch can be seen in Figure 4.

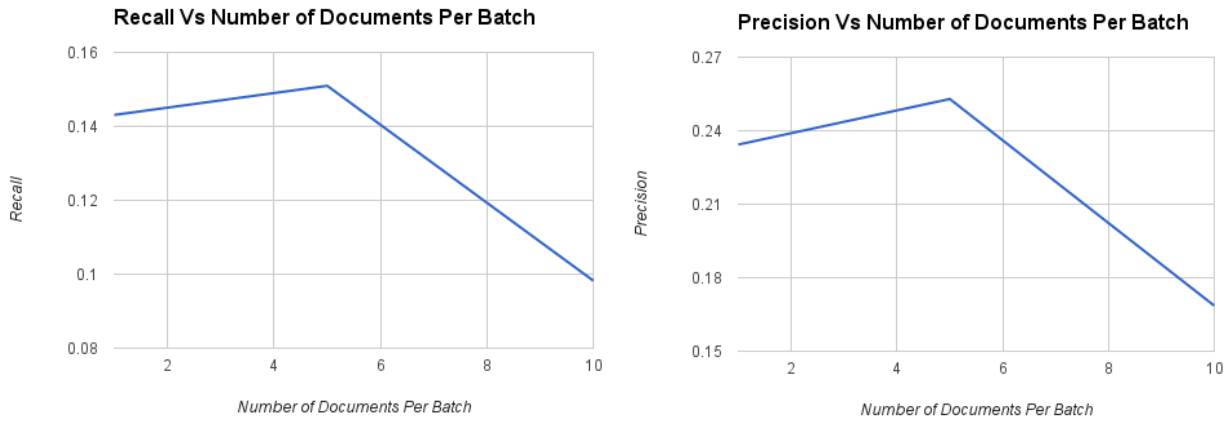

Figure 4: Variation of performance with number of documents per batch

We observe that the performance of MLP is worse than of regression trees as well. Since the performance degrades on increasing the training set size too much, possibly because of overfitting, we did not attempt to learn using larger training set sizes. Linear function approximation seems to have yielded much better results.

6 Connection to Metric Learning

When the chosen value function is a linear function approximation, this process can be interpreted in the following manner. Each mention has a feature representation in a suitable space, and the reinforcement learning algorithm is being used to learn weights of different features such that single link clustering using the resultant weighted distance in this space results in the optimal coreference clustering.

This raises a question as to whether other metric learning schemes can be applied to solve this problem. A number of state-of-the-art metric learning techniques cast the metric learning problem as an optimization problem [6, 7]. However, this is not necessarily the best approach when it is not clear that optimizing metrics such as a suitable squared error results in a good clustering, that is, when the link between the features known and the goodness of the clustering is not clear [1], which is the case in the problem of coreference resolution. Further, such methods typically exploit only pairwise similarity or dissimilarity information. When a complete clustering is available for training, it may be desirable to make use of it directly as there is a loss of information when it is converted to pairwise relations.

As a solution to this, Bagherjeiran *et al.* [1] propose an alternative method called adaptive clustering that uses reinforcement learning to determine a policy for clustering, in terms of feature weights. However, their method requires a full re-clustering of the data after each iteration, which can be expensive. Our method differs from this in the sense that actions only update the current clustering by a single link, whose effect on the goodness of the overall clustering is returned as a reward. This reward updates features weights using standard reinforcement learning update rules, which can then be used to determine the next link to be made. Some directions to explore in this method include the choice of metric used to compare clusterings and the choice of reinforcement learning algorithm employed for the weight updates.

7 Future Work

We have demonstrated how a simple RL model using SARSA(λ) and linear function approximation can perform well for coreference resolution, outperforming existing systems in terms of MUC-Precision. It is possible that the use of alternate function approximations that better capture the true Q-values may perform better on this task.

Further, it is important to note that this method is not specific to coreference resolution and can be applied to any problem that requires adaptive clustering - learning of a clustering when a complete clustering is available as supervision. Some such applications include interactive clustering for summary generation, clustering for streaming data and multi-agent coordination and control [1]. It can also potentially be extended to other structured prediction problems where current solution approaches break down available training data in smaller units in a lossy manner.

References

[1] Abraham Bagherjeiran, Christoph F. Eick, and Ricardo Vilalta. Adaptive clustering: Better representatives with reinforcement learning. Technical Report UH-CS-05-06, University of Houston, 2005.

[2] Pradheep Elango. Coreference resolution: A survey. Technical report, University of Wisconsin Madison.

[3] Veselin Stoyanov, Claire Cardie, Nathan Gilbert, Ellen Riloff, David Buttler, and David Hysom. Reconcile: A coreference resolution research platform. Technical report, Cornell University, 2010.

[4] Veselin Stoyanov and Jason Eisner. Easy-first coreference resolution. In *Proceedings of COLING 2012: Technical Papers*, COLING '12, pages 2519–2534, 2012.

[5] Silke Wagner and Dorothea Wagner. Comparing Clusterings – An Overview. Technical Report 2006-04, Universität Karlsruhe (TH), 2007.

[6] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart J. Russell. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 505–512, 2002.

[7] Yiming Ying and Peng Li. Distance metric learning with eigenvalue optimization. *JMLR*, 13:1–26, 1 2012.

Appendix A - Properties of mentions

Property Name	**Description**
ProperNoun	Is the noun phrase a proper noun
SubsumesNumber	Is the phase a number (only numerical characters)
Embedded	A noun phrase is embedded if it is completely contained in another noun phrase
SoonWords	Content words in the noun phrase
Gender	Associated gender of the noun phrase, if applicable
HeadNoun	The head noun of the phrase, that is, the noun that determines the syntactic type of the phrase
Definite	Checks whether the phrase is definite, that is, starts with the word "the"
ContainsProperName	Does the phrase contain a proper name
Synsets	WordNet synsets corresponding to words in the phrase
Number	Singular or plural, if applicable
Pronoun	Is the phrase a pronoun
InfWords	Infinitive words in the phrase
NPSemanticType	Semantic class such as person, organization, date etc
Animacy	Does this refer to an animate or inanimate object
ProperName	Checks whether the object is a person, organization or location
SentNum	Sentence umber in which the phrase is present
Stopword	Stop words in the phrase
ParNum	Paragraph number in which the phrase is present
WNSemClass	Semantic classes of words in the phrase, according to WordNet
Conjunction	Checks for the presence of a conjunction
GramRole	Grammatical role of the noun phrase
ContainsAcronym	Does the phrase contain an acronym
AllGramRole	Grammatical role of the head noun
ProperNameType	If it is a proper name, whether it is a person, location or organization
Modifier	Obtains the set of adjectives and adverbs

Appendix B - Features of pairs of mentions

Feature Name	Description
Agreement	The value of this feature is C if the NPs agree on both number and gender. If they disagree on either number or gender, the value of the feature is I. The value of the feature is NA if no gender or number information is present for one or both of the NPs in question.
Alias	C if one NP is an alias of the other, otherwise I. Alias can mean a variety of things, such as different was of representing the same date, monetary value or number. This feature uses information about the semantic type of the NPs.
AlwaysCompatible	Always returns C.
Animacy	If the NPs agree on animacy, returns C, else returns I. Uses semantic type information.
Appositive	Return C if the NPs are in an appositive construction, else returns I. Uses semantic type information.
Binding	Returns C if the NPs do not violate conditions B and C in Chomsky's binding theory, else I. In short, element α binds elements β if and only if α c-commands β. A node X c-commands node Y if and only if X does not dominate Y and Y does not dominate X
BothEmbedded	Returns C if both NPs are embedded, NA if only one is, and I if neither is embedded.
BothPronouns	Returns C if both NPs are pronouns, NA if only one is, and I if neither is a pronoun.
BothProperNouns	Returns C if both NPs are proper nouns, NA if only one is, and I if neither is a proper noun.
BothSubjects	Returns C if both NPs are in the subject position relative to a verb clause, NA if only one is, and I if neither is a subject.
ConsecutiveSentences	Returns C if the NPs are in consecutive sentences, otherwise returns I.
Constraints	Checks the compatibility of the GENDER, NUMBER, CONTRAINDICES, ANIMACY, PRONOUN and CONTAINSPN, if all are compatible (or at least not Incompatible types for the last four features), then it returns C, else I.
ContainsPN	This features checks that both NPs contain proper names and contain no words in common, if this is true, it returns I, else C.
Contraindices	The following constraints are implemented for this feature: (1) two NP's separated by a preposition cannot be coindexed and (2) two non pronominal NP's separated by a non-copular verb cannot be co-indexed. If the two NP's violate these conditions, then returns I, else it is C.
Definite1	Returns Y if the first NP starts with 'the', else N.
Definite2	Returns Y if the second NP starts with 'the', else N.
Demonstrative2	Returns Y if the second NP starts with a demonstrative, i.e. this, that, these and those. Returns N otherwise.
DocNo	A bookkeeping (uninformative) feature - the internal number of the document from which the NPs came from.
Embedded1	Y if the first NP is an embedded or nested NP, else N.
Embedded2	Y if the second NP is an embedded or nested NP, else N.

Gender	If the two NPs agree in gender, then this feature returns C, and I if they disagree. If the gender information is not determined by the system, then it returns NA.
HeadMatch	Checks for matching head noun between the two NPs, if they match, returns C, else I.
IAntes	Returns Y if one of the two NPs is the pronoun I and the other NP is determined to be the quoted speaker of the text containing the I pronoun by a rule based system.
ID1	The identification number of the first NP - another bookkeeping feature and is not used in training nor testing.
ID2	The identification number of the second NP.
Indefinite	I if the second NP is an indefinite and is not an appositive, C otherwise.
Modifier	If the prenominal modifiers of one np are a subset of the prenominal modifiers of the other nps, then returns C, else I.
Number	If the two NPs agree in number, then this feature returns C, and I if they disagree. If the number information of one or more of the NPs cannot be determined, the value is NA.
PairType	Encodes the type of the np pair, i.e., if the pair are Proper nouns, pronouns, definite or indefinite.
ParNum	The distance between the two NPs as they occur in the text in terms of paragraphs.
PNStr	If both NPs are proper names and the same string, then C, else I.
PNSubstr	If both NPs are proper names and one is a substring of the other, then C, else I.
Prednom	If the NPs form a predicate nominal construction, then C, else I. An example: *Barack Obama is the U.S. president.*, *the U.S. President* is acting as the predicate nominal.
ProComp	If both NPs are pronouns and are compatible in gender, number and person, (i.e., he and his), then C, otherwise I.
Pronoun1	If NP1 is a pronoun, return Y, else N.
Pronoun2	If NP2 is a pronoun, return Y, else N.
Pronoun	If NP1 is a pronoun and NP2 is not, then return I, else C.
ProperName	If both NPs are proper names and share no words in common, then returns I, else C.
ProperNoun	If both NPs are proper nouns and share no words in common, then returns I, else C.
ProResolve	If one NP is a pronoun and the other NP is its antecedent according to a rule-based algorithm, then C, else I.
ProStr	Return C if both NPs are pronouns and their strings match exactly, otherwise I.
Quantity	Returns C if the two NPs form the pattern - *sum* of *money* (e.g. loss of 1 million), else I.
SameParagraph	The NPs are found in the same paragraph, then return Y, else N.
SameSentence	The NPs are found in the same sentence, returns Y, else N.
SentNum	The distance between the two NPs in terms of sentences.
SoonStr	If after discarding uninformative words, the strings two NPs match, then return C, else I.
Span	Returns I if one NP spans the other, else C.

Subclass	If one NP's WordNet class is a subclass of the other NP return Y else N.
Subject1	Returns Y if NP1 is a subject, otherwise N.
Subject2	Returns Y if NP2 is a subject, otherwise N.
Syntax	If the two NP's have incompatible values for BINDING, CONTRAINDICES, SPAN, or MAXIMALNP, then returns I, else C.
WNSynonyms	Returns C if the NPs are WordNet synonyms, else I.
WordNetClass	Returns C if both NPs have the same WordNet class, else I.
WordNetDist	The distance in the WordNet Synset tree between the two NPs.
WordNetSense	Returns the first WordNet sense that both NPs share.
WordOverlap	If the intersection of the content words of the two nps is not empty, then C, else I.
WordsStr	If both NPs are non-pronominal and the strings match, then C, else I.
WordsSubstr	If both NPs are non-pronominal and one np is proper substrings of the other with respect to content words, then returns C, else I.