Evaluating Name-Matching for Coreference Resolution

Olga Uryupina

Computational Linguistics, Saarland University
66041 Saarbrücken, Germany
ourioupi@coli.uni-sb.de

Abstract

In this paper we describe experiments aimed at improving matching techniques for the Coreference Resolution task. Combining algorithms proposed in the literature with our own solutions, we run machine learning experiments and evaluate extensively different features and feature combinations to find the best settings. These settings do not only show good performance for English data, but should also be easily adjustable to cover other languages.

1. Introduction

Robust and accurate Coreference Resolution (CR) is essential for many Natural Language Processing tasks, for example, Question Answering. However, not all types of anaphors can be successfully resolved by state-of-the-art CR systems. Typical pronoun resolution algorithms have an accuracy of up to $85 - 90\%$ (Mitkov, 1999) whereas the performance of the best general coreference resolution systems is much lower: for example, (Soon et al., 2001) reported 60.4% F-score on the MUC-7 test data.

This leads to the conclusion that one should try to improve the resolution for the other major types of anaphors: proper names and definite noun phrases. Our experiments address a problem of coreference for named entities, in particular, anaphoric links between two proper names. Table 1 shows several examples of coreferring proper names, demonstrating typical problems for automatic resolution.

There are several reasons to pay attention to this particular sub-task. First, proper names represent a high proportion of all the anaphors: identifying correctly all the anaphoric links between NEs, we achieve 30% recall for the NP-Coreference task (only full NPs are considered as markables) in the set-based evaluation (cf. Section 6.2.).

Second, coreference resolution engines are usually not independent tools — they are integrated into various NLP applications. For example, a good CR module in a question answering system would allow us to keep track of entities, collecting and combining information from different sentences of the text. In this context coreference links between named entities become very important: they bring together various facts about real world objects that are likely to be mentioned in user's questions.

Third, accurate name matching algorithms can be helpful for other tasks. For example, Branting (2002) points out that they are crucial for Legal Case-Management Systems.

In this paper we present various machine learning experiments evaluating strategies for name matching.

2. Previous Studies

Virtually none of the previous approaches to the coreference resolution task include any specific machine learning technique for resolving proper name anaphoric links.[1] The most commonly used matching features for CR in general are *same surface* , *same head*, and *contain* (an anaphor is a substring of an antecedent). In our experiments we use the *same surface* matching as a baseline.

Antecedent	Anaphor
MCDONALD'S	McDonald's
The Federal Aviation Administration	FAA
F-14	F14
CHINA'S Foreign Trade Minister Wu Yi	Ms Wu

Table 1: Examples of problematic cases for proper names coreference resolution

Soon et al. (2001) introduce the *weak string identity* feature (`true` if an anaphor and an antecedent have the same surface form after stripping off the determiners).

Strube et al. (2002) propose two minimum edit distance-based (MED) features. Although useful for German, they might be not so helpful for English because of its simpler morphology. To evaluate the role of approximate matching, we implemented a family of MED-based features.

Branting (2002) investigates name matching strategies for Legal Case-Management Systems, proposing techniques for abstracting over NPs and efficient indexing. Although we do some normalisation, following the ideas of (Branting, 2002) and (Soon et al., 2001), we are not interested in further generalisation over the markables. Branting also uses exact and approximate matching and precompiled abbreviation tables. We do not use any external abbreviation data, but compute possible abbreviations dynamically.

To sum it up, the commonly used approach is very simple, and various previous studies show that it can be improved in many different ways. In this work we combine the ideas proposed in the literature with our own matching strategies.

3. Data

For our experiments we use the MUC-7 coreference corpus (Hirschman and Chinchor, 1997). We parsed the texts with the (Charniak, 2000) parser and collected all the NPs,[2] discarding complex ones (containing an embedded

[1] An exception is a system described in (Strube et al., 2002), where two matching strategies have been tested on named entities separately from all the other types of anaphors.

[2] The MUC-7 scheme covers more types of markables, for example, posessive pronouns

NP). Our estimations show that a system, correctly resolving all the NPs, would achieve $75-80\%$ recall and 100% precision on the MUC-7 test data in set-based evaluation.

We only analyse proper names (NPs with NNP-, NNPS-, or CD-tagged heads). A system, correctly resolving all those NPs, would have 31.7% recall and 100% precision when evaluated against the NPs part of the MUC-7 data.

In our experiments we use the rule induction learner Ripper (Cohen, 1995). To produce data instances, we pair the NPs with each other (every NP was paired with all the preceding ones). This resulted in 20826 (942 positive and 19884 negative) items for the "dryrun" subcorpus and 18702 (851 positive and 17851 negative) items for the "testing" subcorpus.

4. Computing similarity between proper names

We decompose our problem into three major sub-tasks: *normalisation*, *substrings selection*, and *matching*: one can, for example, compute minimum edit distance (*matching*) between the down-cased (*normalisation*) last nouns (*substring*) of an anaphor and an antecedent. The resulting value can be used as a similarity measure between the two.

Below we describe the algorithms we implemented to tackle these three sub-tasks. Several techniques are language specific, whereas others are relatively language independent (possibly applicable to any language with an alphabetic script). Some algorithms are very fast, some (parser-based) require more processing, and some (Internet-based) are very time-consuming. In Section 6. we evaluate "language independent" and "fast" settings separately.

Normalisation. We use three normalisation functions: *no_case*, *no_punctuation*, and *no_determiner*. The first one transforms a string into lower-case format. The second one strips off all the punctuation and auxiliary characters, and the last one strips off the determiner. The first two normalisation techniques are relatively language independent. The third one is obviously language dependent: we need a list of all the determiners to perform this operation. However, we believe that such a list can be compiled very quickly for any particular language. One can combine these functions sequentially, producing complex normalising algorithms. For example, *no_case&no_punctuation* of "that F-14" is "that f14". Finally, one can use several normalisations for the same markable to compute values for different features.

Substring selection. Some words in a name are more informative than others. Therefore it can be reasonable to compare not the whole strings, but only the most representative parts of them. We implemented several algorithms for selecting most informative words.

The *last_noun* algorithm outputs the last noun of the NP string. It requires a parser (or at least a tagger) and is language-dependent and moderately time-consuming.

Last is a simple modification of the *last_noun* algorithm. It outputs the last word of the NP string. Although less accurate, it is faster and language independent.

First, a counterpart of *last*, is the first word of the NP.

The *rarest* algorithm outputs the least frequent word of the NP string: each word is sent to the AltaVista search engine and then the one that gets the lowest count (number of pages worldwide written in English) is returned. This algorithm is language independent, but very time-consuming.

matching	formula	value
MED_s	MED in symbols	4
MED_s_anaph	$\frac{MED\_s}{length\_s(anaphor)}$	1
MED_s_ante	$\frac{MED\_s}{length\_s(antecedent)}$	0.5
MED_w	MED in words	1
MED_w_anaph	$\frac{MED\_w}{length\_w(anaphor)}$	1
MED_w_ante	$\frac{MED\_w}{length\_w(antecedent)}$	0.5

Table 2: Approximate matching functions and their values for (*New York*, *York*).

As an example, the *last_noun* and *last* substring of "Lockheed Martin Corp." is "Corp.", whereas the *first* and *rarest* substring is "Lockheed".

It does not make any sense to combine these algorithms sequentially, as they always output one word. However, one can use several of them at the same time: for example, our *fast* configuration uses both the *first* and the *last* algorithms.

Matching. We investigated the following string matching algorithms.

exact_match is a boolean function on two strings. It outputs 1 if they are identical and 0 otherwise.

approximate_match: we implemented a family of algorithms, based on the minimum edit distance measure (Wagner and Fischer, 1974). Given two ordered sequences, the MED between them is defined as the number of insertions, deletions, and substitutions, needed to transform one into the other. We measure the distance between two strings either in symbols (*MED_s*) or in words (*MED_w*). In addition to the bare counts, we normalise our MED values by the length of an anaphor or an antecedent in symbols (*length_s*) or in words (*length_w*). The formulas are shown in Table 2.

matched_part algorithms are generalisations of commonly used *contain* feature (cf. Section 2.). They represent the size of the overlap between two NPs. The basic *matched_part* algorithm computes the number of symbols/words two NPs share. We normalise this count by the length of an anaphor or an antecedent in the same way as it is done for the approximate matching.

abbreviation: we have implemented several methods to resolve abbreviations, all of them comparing an anaphor (full string) to the *last_noun* of candidate antecedents . The first algorithm (*abbrev1*) takes the initial letter of all the words in a string and produces a word out of them. This word is compared (by exact match) to the head of the second NP. The second algorithm (*abbrev2*) does the same, but ignores words, beginning with low-case letters. The algorithm *abbrev3* checks whether it is possible to split the head of the second NP into small units, such that each unit is a beginning (prefix) of a word in the first NP, and the prefixes come in the right order (the same as the order of the corresponding words). Finally, *abbrev4* proceeds in the same way, but does not allow empty prefixes. The first two algorithms represent the most commonly used abbreviations. The last two algorithms are more general, allowing for non-trivial ways of abbreviating. Table 3 shows some examples.

Again, one cannot combine these algorithms sequentially, but it is possible to use several of them at the same

(antecedent, anaphor) pair	abbrev1	abbrev2	abbrev3	abbrev4
(United States, U.S.)	1	1	1	1
(The Federal Bureau of Investigations, FBI)	0	1	1	0
(The Federal Bureau of Investigations, Bureau)	0	0	1	0
(Silicon, SILIC)	0	0	1	1

Table 3: Example values of the abbreviation functions (assuming full normalization).

np_type	NP's article (the, a(n), none)
number	number of the head noun (sg, pl, none)
premodifiers	NP contains premodifiers (yes, no)
postmodifiers	NP contains postmodifiers (yes, no)
appositive	NP is a part of an appositive construction (first_part, second_part, no)
conjunction	NP is a conjunction (yes, no)
rarest	the AltaVista count for the *rarest* word (1...n)
length_s	length of the NP string in symbols (1...n)
length_w	length of the NP string in words (1...n)

Table 4: NP-based features

time, computing values for different features.

5. Features and Feature Configurations

All the techniques described above help us to build a feature set. We divide our features into two groups.

NP-based features, computed for both an anaphor and an antecedent, represent information about a single noun phrase. The features are represented in Table 4. However, we do not use all of these features in all the experiments.

Link-based features describe the similarity between two NP strings. The only exceptions are the *sister* and *distance* features. The *sister* feature is set to 1 when two NPs are sister nodes in a parse tree and to 0 otherwise. Two *distance* features encode the distance between two named entities measured in terms of NPs and sentences.

The other features are represented by triples (*normalisation*, *substring selection*, *matching*). Not all of them are useful. For example, the *matched_part* algorithm with any substring selection would always produce the same values as *exact_match* with this substring. This observation reduces the total number of features dramatically. Overall we have 135 features: 18 NP-based ones, *sister*, 2 *distances*, and 114 features represented by matching triples.

5.1. Configurations

We want to test such hypotheses as, for example, "Approximate matching yields better results than exact matching", comparing the results in the cases when only several features are used. We call these groups of features *configurations*. Below we describe some of our configurations:

all: all the features.

baseline: all the NP-based features, *sister*, *distance*, exact matching for full NPs, no normalisation

last_noun: all the baseline features, all the *(__,last_noun, exact_match)* triples.

fast: all the features that do not require Internet counts

faster: all the features that require neither Internet counts nor parsing (i.e., all the types of matching for full NP strings and their *first* and *last* substrings)

MED_only: all the NP-based features, *sister*, *distance*, and all the *(__,__, approximate_match)* triples.

MED_surface_only: all the NP-based features, *sister*, *distance*, all the *(__,no_substring_selection, approximate_match)* triples.

last, *first*, *rarest*: *rarest_count* (for the *rarest* configuration only), *length*, *distance*, all the *(__, no_substring_selection, exact_match)* and *(__, sub, exact_match)* triples; *sub* is *last*,... correspondingly.

We evaluated 29 different configurations, but, due to the lack of space, we describe only the most interesting ones.

6. Evaluation

6.1. Cross-validation of Ripper's classifiers

In this experiment we used only the MUC-7 dry-run data (30 texts). The experiment was organised as follows. For each of our 10 cross-validation runs, we reserved 3 texts for testing. The remaining texts were first used to optimise Ripper's *S* parameter (degree of hypothesis simplification) by 3-fold cross validation. Finally, we trained Ripper on all the 27 texts with the best *S* value and test on the reserved 3 texts. The performance was measured in the standard way (precision is a ratio of correctly classified links over all the anaphoric links suggested by Ripper and so on).

The results are presented in Table 5: the upper part shows the configurations, requiring neither a parser, nor web counts, the middle part — the ones, requiring only a parser, and the lower part — the ones, requiring web counts. All the configurations performed significantly ($p < 0.01$, two-tailed t-test) better than the baseline.

6.2. MUC-style set-based evaluation.

In this experiment we used the dry-run data for training and 20 MUC-7 test texts for testing. The MUC scoring program (Vilain et al., 1995) was used for evaluation. The scorer compares not individual pairs, but whole coreference chains for estimating the system's performance. So, we implemented the same resolution algorithm, as the one proposed in (Strube et al., 2002) and (Ng and Cardie, 2002). For each anaphor, we check the candidate antecedents (preceding proper names), starting with the closest one and proceeding backwards. We submit all the pairs to Ripper one by one. If a pair is classified as a possible coreference

	normalisation		
	no	full	all features
no parsing, no web counts			
baseline	57.8	(63.9)	(63.5)
first	62.7	73.5	72.4
last	67.7	71.1	71.9
faster	79.6	82.4	81.2
MED_surface_only	75.2	82.8	80.4
parsing, no web counts			
last_noun	78.2	80.2	81.3
MED_only	82.1	83.0	83.8
fast	81.4	83.6	83.5
web counts			
rarest	78.2	80.3	73.1
all	82.5	82.5	82.1

Table 5: The system's performance (F-measure) in the 10-fold cross-validation on the MUC-7 dry-run data.

	normalisation		
	no	full	all features
no parsing, no web counts			
baseline	71.7	(72.7)	(71.9)
first	72.4	77.7	75.2
last	71.8	75.9	71.5
faster	75.9	75.3	80.1
MED_surface_only	77.9	79.3	80.5
parsing, no web counts			
last_noun	79.4	80.6	79.4
MED_only	80.0	79.2	77.7
fast	77.3	78.6	80.4
web counts			
rarest	77.6	75.7	76.9
all	77.6	82.5	78.2

Table 6: The system's performance (F-measure) in the set evaluation on the MUC-7 test data.

link, we add the anaphor to the coreference chain of the antecedent and proceed to the next anaphor.

The evaluation results are shown in Table 6. Our baseline has a relatively high precision (82.6%), but very low recall (61.1%). All the other configurations, except *first* and *last* have slightly lower precision, but significantly ($p < 0.05$, χ^2-test) higher recall. We cannot compare the corresponding F-scores, as the χ^2-test is not applicable.

7. Discussion and Conclusion.

We decomposed the NE coreference problem into three sub-tasks: normalisation, substring selection, and matching. Combining different solutions to these sub-tasks we came up with several feature configurations to be evaluated.

As our experiments show, sophisticated matching algorithms clearly outperform the baseline: the best configuration in our first experiment yields an error reduction of 61%. However, tuning these algorithms to achieve the best performance is not a trivial task.

The substring selection is useful provided it is done properly, as in the *last_noun* configuration. The Internet-based substring selection (*rarest*) is only slightly worse. Unfortunately, similar techniques (*first* and *last*) can bring only moderate advantage over the baseline. So, if we want to investigate the CR task in other languages, where parsing resources are less reliable or even non-existent, we should try another solution, instead of the substring selection: either use several substrings at the same time (*faster*), or improve the other parts of our matching algorithms (MED).

All the sophisticated matching functions improve the performance to some extent, although *abbreviations* seem to be almost useless. The most important function is *approximate matching*: with the MED features activated, all the additional improvements do not affect the performance significantly, consider the differences in F-measure for *MED_only* (with the *last_noun* substring selection) and *MED_surface_only* (without any substring selection).

Finally, we could not find a normalisation function outperforming all the others in all the cases. But the experiments show that it is worth using at least some normalisation: in almost all the configurations *no normalisation* results in a significant drop of the performance. With *approximate matching*, the normalisation choice does not play an important role ($2-3\%$ difference in F-measure in the first experiment, except for the *no normalisation* case). With *exact matching*, normalisation becomes more important (up to 10% difference in the F-measure).

In future we plan to follow two directions. First, we want to apply our techniques to all the markables (not only named entities as in present study). Second, we plan to use the same algorithms for the CR task in another languages. Finally, once we are able to resolve links between two proper names, we want to investigate the coreference between proper and common nouns.

8. References

Branting, L. Karl, 2002. Name-matching algorithms for legal case-management systems. *Journal of Information, Law & Technology*, (1).

Charniak, Eugene, 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*.

Cohen, William W., 1995. Fast effective rule induction. In *Proceedings of ICML*.

Hirschman, Lynette and Nancy Chinchor, 1997. Muc-7 coreference task definition. In *Proceedings of MUC-7*.

Mitkov, Ruslan, 1999. Anaphora resolution: the state of the art. Technical report, University of Wolverhampton.

Ng, Vincent and Claire Cardie, 2002. Combining sample selection and error-driven pruning for machine learning of coreference rules. In *Proceedings of EMNLP*.

Soon, Wee Meng, Hwee Tou Ng, and Daniel Chung Yong Lim, 2001. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.

Strube, Michael, Stefan Rapp, and Christof Müller, 2002. The influence of minimum edit distance on reference resolution. In *Proceedings of EMNLP*.

Vilain, Marc, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman, 1995. A model-theoretic coreference scoring scheme. In *Proceedings of MUC-6*.

Wagner, Robert A. and Michael J. Fischer, 1974. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173.