A Machine Learning Approach for Coreference Resolution

Maschinelle Lernverfahren zur Koreferenz Resolution
Master-Thesis by Thomas Arnold from Lindenfels
Date of Submission:

1. Referee: Prof. Dr. Karsten Weihe
2. Referee: Prof. Dr. Johannes Fürnkranz
3. Referee: Prof. Dr. Anette Frank

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Department of Algorithmics

A Machine Learning Approach for Coreference Resolution
Maschinelle Lernverfahren zur Koreferenz Resolution

Submitted Master-Thesis by Thomas Arnold from Lindenfels

1. Referee: Prof. Dr. Karsten Weihe
2. Referee: Prof. Dr. Johannes Fürnkranz
3. Referee: Prof. Dr. Anette Frank

Date of submission:

Technische Universität Darmstadt
Department of Algorithmics
Department of Computer Science

Prof. Dr. Karsten Weihe

Authors Declaration

This Master thesis has been carried out in the Department of Algorithmics, Technische Universität Darmstadt, Germany, under the guidance of Prof. Dr. Karsten Weihe and Prof. Dr. Johannes Fürnkranz with additional guidance from Prof. Dr. Anette Frank from Heidelberg University.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, 09. December 2014 Thomas Arnold

Contents

Abstract

In order to research machine learning techniques on natural language data, two different machine learning algorithm using decision trees were applied to the gold annotated English Ontonotes Corpus for coreference resolution. Positive and negative instances were built with a fixed maximum distance parameter, so the distance could be used in the feature set along with various natural language attributes and added identity features. The first algorithm, J48, had huge runtime problems and could not be enhanced to acceptable performance. The second algorithm, Random Forest, has not been explored much for coreference resolution yet. It handled the big amount of data and features much better, so various experiments were performed to improve the results. The predictions were transformed to ranking results and combined with Stanford CoreNLP in the experiment scenario, obtaining slightly improved predictions. Additionally, a first approach using motive based features was developed that uses reoccurring patterns as a probability feature.

1 Introduction

A news article may contain the following line: ”He was sentenced to five years in prison for this crime.” Maybe this line has the most important piece of information of the whole article, but without additional context, is is almost useless for the reader. The pronoun ”he” refers to a person that has to be introduced first, and the criminal activity that is denoted as ”this crime” is also unknown. This additional information can most likely be found in on of the preceding sentences: ”John Doe was arrested for bank robbery and brought to court today.” This sentence contains the needed clues to link the pronoun ”He” to the name ”John Doe”, as they refer to the same entity, and explains that ”this crime” has been a bank heist. Therefore, both sentences are needed to extract the core information of the article, but this may not always be desirable. To generate a very short summary of the most important information, the nominal phrases of the latter sentence have to be connected to the corresponding phrases of the preceding sentence: ”John Doe was sentenced to five years prison for bank robbery.”
Noun phrase Coreference Resolution is a difficult Natural Language Processing task. According to Vincent Ng [8], it is followed extensively since the 1960s. The main issue is to find pairs of nominal phrases that refer to the same entity. The mentioned example can be solved with relatively simple approaches, as ”John Doe” can only be coreferent to the pronoun ”He”, and not to ”this crime”. Generally, this is a very challenging problem, as the current role of the used nominal phrases can be difficult to determine, or external knowledge has to be used.
To tackle this problem, supervised machine learning approaches were used. The core purpose of machine learning is to generate knowledge from experience. In this work, the ”experience” were annotated texts with manually determined coreferences and several other lexical properties, and the desired ”knowledge” was the ability to find coreferencial nominal phrases in unknown texts. The first research task was to determine if this approach is successful, and how well coreferences can be learned from lexical features. Additionally, the impact of the different features on the results will be investigated, as well as the performance of the used machine learning algorithms.
As coreference resolution is a very complex task, this approach was not primarily aimed to create a system that can solve this problem on its own on a competitive level. For comparison, the resulting approaches were combined with the existing Natural Language Processing Toolkit Stanford CoreNLP to determine if they can improve its predictions. Additionally, first steps of a motive based approach were performed that created a very simple frequency based prediction feature.

The core research questions of this work were formulated in the following way:

- Can coreferences be predicted using lexical features?
- Which approach appears to be more / less successful?
- Which features have high / low impact on coreference?
- Can this approach improve an existing coreference resolution system?

This work was created in cooperation with the department of Computational Linguistics at Heidelberg University that provided both support at linguistic topics and all input data in a customized format. Additionally, the department contributed guidance at formalising and constructing the machine learning experiments including evaluation, and advice at constructing a motive based approach.

The following chapter 2 provides essential basic information about coreference resolution and machine learning that is needed for understanding the specific implementation, used features and conducted experiments that are explained in chapter 3. All experimental results and their interpretation can be found in chapter 4, and this work closes with final conclusions and an outlook in chapter 5.

2 Theoretical basics

This chapter contains brief explanations of all concepts of the implemented approach. First, the core problem is described by introducing coreferences and coreference resolution. Then, the general idea of machine learning and decision tree learning is explained and applied to the problem of coreference resolution. Finally, the used machine learning algorithms are presented.

2.1 Coreference

In proper natural language, single entities have to be mentioned multiple times in the same sentence or following sentences. This is very useful to structure information, and necessary to connect different pieces of information to the same person, location or item. Using the full form to describe the relevant entity is very bad form, as it gets repetitive, unnatural and boring to read very quickly: ”Michael leaves the house and enters Michael’s car. Michael drives fast so Michael will not miss Michael’s appointment with Michael’s doctor.” This effect gets even worse when the used entity has a very long name or description that has to be repeated over and over again. For a human reader, the constant repetition makes the content of the text even confusing, as the recurring mention of the name ”Michael” may imply that there could be many different individuals involved in this scene. For this reason, it is both convenient and improves the style tremendously to not use the full form every time, but replace some mentions with pronouns or descriptive terms: ”Michael leaves the house and enters his car. The young student drives very fast so he will not miss his appointment with his doctor.” This form has a much more natural flow, and the pronoun ”he” with the corresponding possessive determiner ”his” make it very clear that the scene contains only one person. Using the phrase ”The young student” as a replacement for the name ”Michael” even introduces additional information about the protagonist without breaking the flow of the sentence or repeating the name.
In this example, ”Michael”, the pronoun ”he” and the phrase ”The young student” have the same referent, as they refer to the same entity, and the phrases are coreferential. These coreferences can appear in many different variants - the full form can precede or follow the abbreviated form, or one pronoun can even combine two separate entities: ”John and Joe run home fast. They are very hungry.” The full form following the pronoun is mostly used in prose texts as a stylistic medium: ”It was his most difficult case, but Detective Johnson had a very strong lead.” Most other forms of language predominantly use the usual, anaphoric form, with the full form in front of the coreferent abbreviation. This is the usual case for structured news articles, as it most easy to read and comprehend.

2.2 Coreference resolution

The goal of an automatic Coreference Resolution system is to find and link all coreferential phrases in an unknown input text. This task may seem easy for a human reader, and in the following case, it can indeed be solved with relatively simple methods: ”Mary meets John. She likes him.” The pronouns ”she” and ”him” have to be connected to the names Mary and John, but since Mary is the only female name, and ”she” is the only female pronoun in this scope, they have to be coreferent. There are much more complicated examples that need additional information about context, current roles of world knowledge to be solvable: ”Johnson had to face Judge Tomson in court today. He was sentenced to five years in prison.” Both mentioned names can possibly be male names, so the easy approach fails here. For a human, this problem is still very easy, as the current roles of the two introduced individuals is very clear here. One of them is the judge, the other the accused culprit. In this scenario, the only rational interpretation is that the culprit is sentenced to prison, not the judge. So the pronoun ”he” has to refer to

the culprit Johnson. This task is much harder to solve for automatic systems, as it requires more complex methods for understanding the current roles of the named entities. The following example even needs up-to-date world knowledge: ”Angela Merkel met Barrack Obama today. The president of the united states prepared a fantastic opening speech for this event.” Only world knowledge allows the reader to know that Barrack Obama is the mentioned president.
To find coreferences between pronouns and nominal phrases in an unknown text, these elements have to be found and determined first, which is done by a process called part-of-speech (POS) tagging. Current POS tagging systems have reported accuracy rating of over 97% on a test corpus, making this one of the more robust and reliable tasks of natural language processing [5]. After the nominal phrases, pronouns and all other components have been identified, the corefering expressions must be found and linked.

2.2.1 Related Work

The summarization of Vincent Ng [8] describes coreference resolution as a central research topic of natural language processing since the 1960s and gives an overview of the most relevant milestones. There have been numerous machine learning approaches in the past years, both for single document references, like Mitkov [6] or Yang et al. [13], and for cross-document coreferencing, as seen from Bagga and Baldwin [1]. The main algorithm of this thesis, Random Forest, has been used on natural language data by Treeratpituk [12] or dos Santos et at. [10], but not as an addition to an already existing system.

2.2.2 Stanford CoreNLP

The Stanford Natural Language Processing tool set ”Stanford CoreNLP” contains algorithms for various natural language processing tasks. Its part-of-speech tagger is one of the best with a reported accuracy rating of 97.32% [5]. Another component of CoreNLP is Coreference Resolution System that uses an ordered sieve structure [4]. At each sieve level, a different approach is used to consider two nominal phrases as coreferent. The upper sieve levels contain more robust approaches, like string matching techniques. If these methods could not determine coreference, the lower sieves use more complex but potentially more error prone approaches, like lexical chain matching using the linguistic resource wordnet.
This Coreference Resolution System is used in this work to test the developed machine learning approach in combination with an established system. The sieve structure can be utilized to insert the developed approach at various sieve levels, but this has not been conquered yet and is considered as a next step. The implemented combination setting is described in chapter 3.6.

2.3 Machine learning

Machine learning is a discipline of computer science that concentrates on methods and algorithms that can accomplish or approve certain tasks by utilizing given data. In most cases, this data is needed as input to construct some kind of model, which then can be used to accomplish the task, or generate some information about new, unseen data. There are many examples for machine learning applications. Many artificial opponents for board games use large amounts of data to learn how to play and win the game. This data may contain session reports of games that famous human players have played, but may also be generated from games that the algorithm plays against itself or other algorithms. In many games like chess or backgammon these approaches have created artificial players that can beat even the worlds best human players. Another example of machine learning applications can be found in recommender systems of shopping websites. With the vast amount of user data and buying records, many of these sites try to guess what the current user may be interested in, based on his purchase history, content of his

shopping cart or currently visited item. The guessed items can then be presented to the user, so he can see and add them to his cart.
The core idea of learning is that knowledge is generated from experience. The generated knowledge differs for every application, and similarly the input used as experience can have many different forms and can be used and presented in many different ways. Machine learning approaches are often differentiated into three major categories:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

In supervised learning scenarios, datasets contain pairs of inputs with corresponding outputs. The learning algorithm tries to learn patterns in a training set of input-output pairs, so it can predict the correct output or class for new, unseen input datasets. To compare and evaluate supervised learners, only a portion of the known, labelled data - the training set - is used for training the algorithm. Another portion of the data, that has not been used for learning, is then used as a test set. The algorithm is used to predict the output class of this data, and the predictions can be compared to the real output to measure the performance of the algorithm. By using the same training and test sets, different methods can be compared with these performance measures. An example for a supervised learning scenario can be seen in spam mail recognition, the e-mails as input data and an output label ”spam / no spam” for every e-mail. A system can be trained on many examples that have been categorized, so it can predict if new e-mails are considered spam or not. Unsupervised learning does only contain input data, with no distinct output label. The goal of an unsupervised learning algorithms changes from prediction to pattern recognition, as the datasets are tried to be clustered or categorized. This can be found in image segmentation techniques that try to group all the pixels of one image into similar regions.
In a reinforcement learning scenario, the algorithm has to archive a certain goal and it knows of its possible actions. By trying these actions and interacting with its environment, the algorithm searches for ways to accomplish the set goal as good as possible. An artificial board game player that plays the game over and over against itself is an example for this scenario. It is restricted by the game rules, and the outcome or current state of the game is the only feedback it gets.

2.3.1 Decision tree learning

Decision tree learning is a category of algorithms to solve supervised machine learning scenarios. Using all the labelled input data, these algorithms try to build a model of one or more decision trees, which are then used to predict the output label of new data examples. Each example is represented by an array of values that are called features. These values may be binary, numerical, nominal, or any other data type. These features may be used to split the training data into many parts in order to separate the data of every possible output type. An example for this idea can be seen in figure 2.1. In this figure, the data consists of different shapes with different properties. Some of them are triangles that may point upwards or not, and all of them can be categorized by their amount of corners. The desired class attribute is the color of the shape, that can be either red or blue. By using some decision tree algorithm, several tests on the different shape features have been determined and consecutively connected, to sort the shapes by color. All tests are applied until the bottom of the tree is reached. These leaf nodes contain the class label that was predominantly observed for all or most of the training examples that followed this exact line of tests, so this class label is predicted for every new example with the same nature. In this example, each triangle that is pointing upwards will be predicted to be red.

The goal of decision tree algorithms is to generalise from the different features to the class label. For this reason, it is not desired to split the tree too often until every training example is separated from each

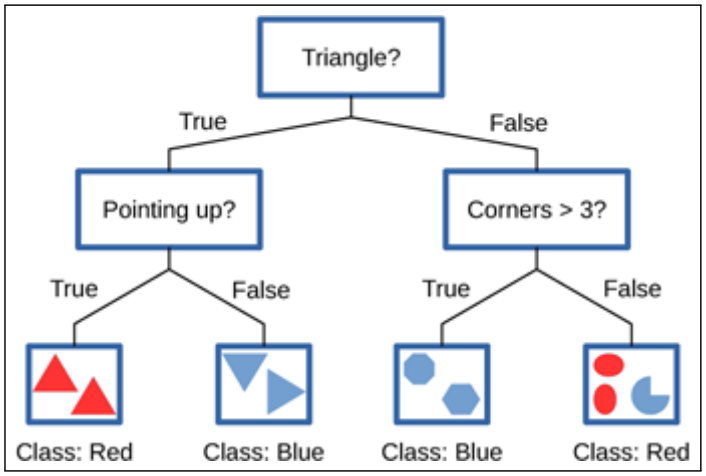

Figure 2.1.: Decision tree example

other. In this case, every training example would be categorized correctly, but the algorithm would not have generated general rules how to categorize new data, instead it tried too hard to match the specific properties of the training set. This problem is known as overfitting, and it has been shown that simpler trees often display much better performance as complicated trees on test sets, even if the performance of the training set is worse.

2.3.2 Coreference resolution with decision trees

As stated in the introduction, these are the core research questions of this work:

- Can coreferences be predicted using lexical features?
- Which approach appears to be more / less successful?
- Which features have high / low impact on coreference?
- Can this approach improve an existing coreference resolution system?

Coreference resolution decision tree algorithms have been briefly explained in the previous chapters. The main idea of this approach is using decision trees to find coreferences in new documents. Large amounts of annotated texts were used as input data, with annotated coreference connections between pairs of nominal phrases as output labels. The used features consist of several linguistic properties of each nominal phrase, like the connected verb or part-of-speech tag. If this idea is successful, a decision tree algorithm can generate a model containing all these different linguistic features that can predict coreferential expressions. The performance of this approach has been evaluated by using a test set and computing appropriate performance measures. Two different decision tree algorithms were compared to determine which one can handle this task more successfully. The impact of the used features was investigated by evaluating the resulting tree models. For the last question, the resulting machine learning system was combined with the Stanford CoreNLP system presented in chapter 2.2.2, and the effect of this combination was examined.

2.4 Used decision tree learning algorithms

Two different machine learning algorithms were used in the conducted experiments. J48 is a java implementation of the algorithm C4.5 that has been used many times many times in natural language processing as in the popular publication by Ng [7] or recent works from Freedman and Krieghbaum

[3], and numerous other machine learning tasks. The second algorithm, Random Forest, was proposed by Breimann [2]. It has only rarely been used in natural language processing tasks like coreference resolution.

2.4.1 J48

C4.5, which J48 is based on, is a machine learning algorithm that constructs a single decision tree. The algorithm was developed by Ross Quinlan [9], and as a standard decision tree algorithm, it has many applications. The algorithm uses a training set containing annotated and classified instances with any number of features and a class attribute. At every construction step, the algorithm selects the feature from the remaining feature set that splits the instances in the best way, which is based on the statistical measure information gain. After the best feature has been determined, all instances get split into two or more smaller subsets using the selected feature as the splitting criterion. This split is represented by a single node in the decision tree that branches out for every resulting split set. If the instances cannot be split again by the used feature, it is removed from the corresponding feature set. This is always the case with binary features, as they only allow for a single ”true / false” split. This procedure is repeated over and over for every subset, until only a single instance is left in a subset or another stopping criterion is met, thus creating a leaf node containing the remaining instances. This creates a decision tree with nodes and leafs. To reduce the amount of overspecialisation and overfitting on the specific training data, various pruning techniques are often applied after the tree has been fully constructed. Several leaf nodes may thereby be combined or intermediate nodes be removed from the tree to make the result more general and more applicable on different data sets. Classification of new instances with the resulting decision tree is very easy. Starting with the first root node, the instance is tested for the feature that was used in the corresponding decision node, and the appropriate branch is selected. There, the following node determines the next test. This is repeated until a final leaf node is reached, and each leaf node determines the classification result of the induced instance depending of the class attribute of the instances that remained there in the training process. This class attribute is predicted for the new instance.

2.4.2 Random Forest

Instead of using a single decision tree Random Forest uses many different decision trees for its predictions. The number of decision trees n can be determined as a variable parameter. For every decision tree, a separate bootstrap sample of the training data instances is created, which is a resampling with replacement. Additionally, at each construction step of the decision trees, the best possible feature is not selected from all remaining features, but from a smaller random subset. The size of this subset is another important degree of freedom when using Random Forest. Each decision tree is constructed separately. To classify a new instance of data, each decision tree gives an individual prediction, and the final prediction is a majority vote of all single predictions. See figure 2.2 for an example of this classification process.

The fact that many different decision trees have to be constructed may suggest that the runtime of Random Forest can be a big problem, but in fact, this algorithms proves to be very fast. Each decision tree only gets a small subsets of instances that have to be considered, and the amount of features that have to be examined at each construction step is also much reduced by the random selection. With this combination, each single decision tree can be constructed much faster than the single decision tree of J48, and since the construction of every tree is completely separate, this process can potentially be parallelized with high efficiency. This is also true for the classification process of new instances, as each single prediction can be made individually. The majority vote at the end of the classification process is simple and fast.

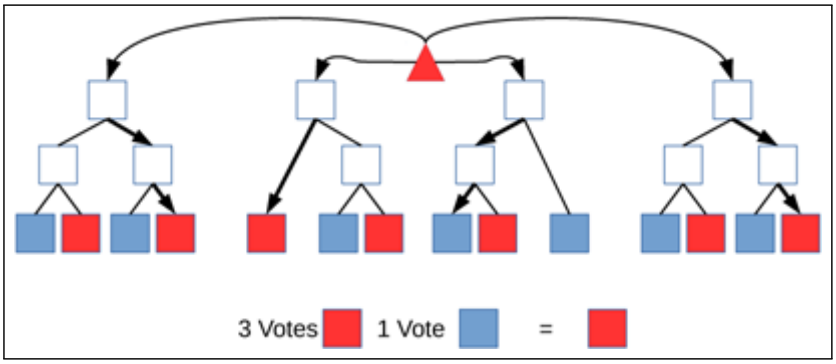

Figure 2.2.: Random Forest example

2.4.3 Evaluation

To evaluate the predictions of a tree learning algorithm, the predictions of the test set were compared to the true class attribute for each instance, counting each different type of hit or error. A positive instance that was predicted to be positive was counted as a ”true positive” (tp), otherwise if it was predicted to be negative it would be a ”false negative” (fn). Negative instances that were correctly predicted to be negative are counted as ”true negatives” (tn), or as ”false positives” (fp) if they were incorrectly labelled to have a positive class attribute. These four values can be used to compute common performance measures. The ratio of true positive predictions versus false positive predictions is called Precision, which reaches its best value at 1.0 and its worst value at 0.0. A Precision value of 1.0 indicates that every instance that was predicted to be positive indeed has a positive class attribute, although there may be any number of positive instances that were missed. This is quantified by the Recall value, that measures the proportion of true instances that were predicted to be positive relative to the true instances that were incorrectly denoted to be negative. The Recall value can also vary between 1.0 and 0.0. A Recall value of 1.0 implies that every positive instance in the test set was correctly predicted to have a positive class attribute, although there may be any number of additional instances that were incorrectly labelled positive, too. The F-Score considers both Precision and Recall and combines them to a single value. An F-Score of 1.0 means that every single prediction is correct, without any errors of any type. The formulas of all used evaluation measures are presented in table 4.1.

Precision	$\frac{TruePositives}{FalsePositives}$
Recall	$\frac{TruePositives}{FalseNegatives}$
F-Score	$2 * \frac{precision * recall}{precision + recall}$

Table 2.1.: Evaluation measures

3 Implementation

This chapter contains details about the implementation of the machine learning approach, beginning with the preprocessing steps. The different experiment set-ups on the various test and training data sets are described, followed by a brief initial solution for motive utilization is presented.

3.1 Input data

Two different corpora were used as resources for experiments or statistics: The English Gigaword corpus and the Ontonotes Release 5 corpus. English Gigaword is a very large collection of english news data collected by Linguistic Data Consortium (LDC). It contains over 4 million documents from different press agencies, and from about 8 years. There are no gold annotations for this corpus, so it was not used as training data for the machine learning process, but the huge amount of data offered many possibilities for large statistical evaluations. The Ontonotes is considerably smaller, containing 3310 documents, but with existing gold annotations for all of them. These annotations include coreferences that can be used as training data for machine learning algorithms.

Both corpora were provided in customized GraphML format by the department of Computational Linguistics at Heidelberg University. This format included all verbs of the documents with possible subject, direct object and indirect object arguments and all relevant properties, and coreference chains. In these chains, every expression that is coreferent to another expression mentioned earlier in the document is connected only to the first coreferent occurrence, which itself is only connected with the next earlier coreferent expression. These coreference chains correspond to the used data model described in the preceding chapter **??**. Details about the used GraphML data format and its attributes are presented in chapter 3.2.

3.2 Preprocessing and instance generation

As described in chapter 3.1, the input data consisted of numerous annotated documents in GraphML format. Several conversion and preparation programs were developed to generate data that could be processed with the used machine learning tools. Additional programs had to be implemented to analyse the resulting predictions and models. This section gives an overview of all these steps from input data to analysis results, and states reasons for every step.

Every document in the corpus pool is represented by its own data file in GraphML format. In this format, every annotated verb of the document is modelled as a node, and every argument of the verb is a port to this node. As an example, the sentence ”Mary leaves the car.” is represented as one node of the verb ”leave” with various attributes, and two ports. The first port corresponds to the grammatical subject of this sentence, ”Mary”, and the second port describes the grammatical object, ”the car”. Both ports also contain several annotated attributes. Each GraphML file then includes coreferencial expressions by defining edges between two ports each. In our example, the sentence ”Mary leaves the car.” is followed by ”Then she enters the house.”. One edge should connect the subject port ”Mary” of the first verb with the subject port ”She” of the second verb to indicate that these two instances refer to the same entity.

In order to use normal machine learning techniques on this data, it had to be transformed into positive and negative instances. In GraphML terms, each instance consists of a pair of ports. If there is an edge that connects these two ports, the corresponding instance is positive. Otherwise, it is a negative instance.

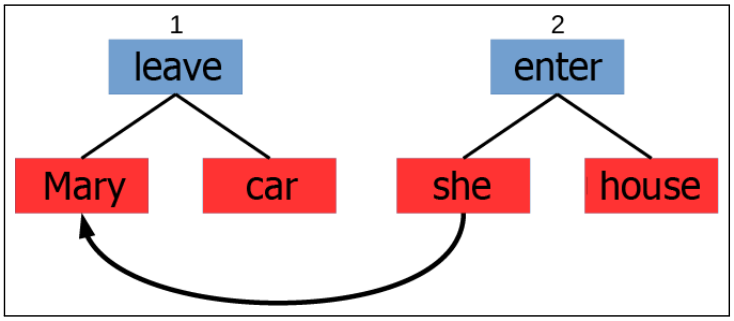

Figure 3.1.: Graph structure derived from the example sentence: ”Mary leaved the car, then she enters the house.”

The edges are directed, and arc from a source port to the next corefering target port. It was determined that all arcs are facing backwards in the flow of the document, so every source port is found later in the text then the target node. There are many ways to determine the possible pairings of these ports, and this is an important degree of freedom in the implementation of this experiment. In theory, every possible combination of all ports from all documents can be transformed into instances, but this would result in enormous amounts of data with an extremely small percentage of positive instances. This leads to very long processing times and very difficult machine learning prerequisites due to the massive dominance of negative instances. The pairings of ports had to be limited. The first limitation is simple and intuitive: Pairings that span two different documents are prohibited - every port can only be paired with ports from its own document. All provided annotations are limited to single documents, so no cross-document coreferences were supported. It would go beyond the scope of this work to add derive these coreferences from the used data, so pairings of ports follow the limits of the annotations, without losing a single positive instance.
Even with this limitation, the amount of instances was still too big to handle properly. To obtain a manageable size, a maximum distance for verbs was introduced. As explained, every verb in a single document is represented by a node in the GraphML file. All nodes from one document are consecutively numbered in order of appearance. The distance of two verbs can now be computed as the difference of its numberings. All ports are connected to a node, so a distance of two ports can be obtained as the distance of the two corresponding nodes. The amount of instances can now be adjusted by limiting the distance of each pairing to a fixed maximum distance. It has to be considered that this method not only reduced the amount of negative instances, but also excludes a portion of the positive instances. To get an estimate of the usual distance between coreferent expressions, statistical analysis of the used Ontonotes database was compiled. With this information, the maximum distance was set to six for all conducted experiments. This includes about 92% of all annotated coreferences, as shown in figure 3.2. The resulting data size ensures a reasonable trade-off between computation times and loss of data.

Another constraint of the used data model directly follows this data reduction. If an unlimited scope for expression pairs was used, a possible task would be to find all coreferences for every single expression. By limiting the maximum distance, this task is now impossible, because one unique entity can appear throughout an entire document, which exceeds the set limits. But in this scenario, chances are very high that these limits contain at least one coreferent expression. From there, another coreference may be found within this maximum distance. This forms a chain of coreferent expressions that all belong to the same entity, as shown in figure **??**. It is no longer needed to find all coreferences of a single expression, it is enough to only find the nearest coreference in these chains. This was decided to be the task of the machine learning algorithms.

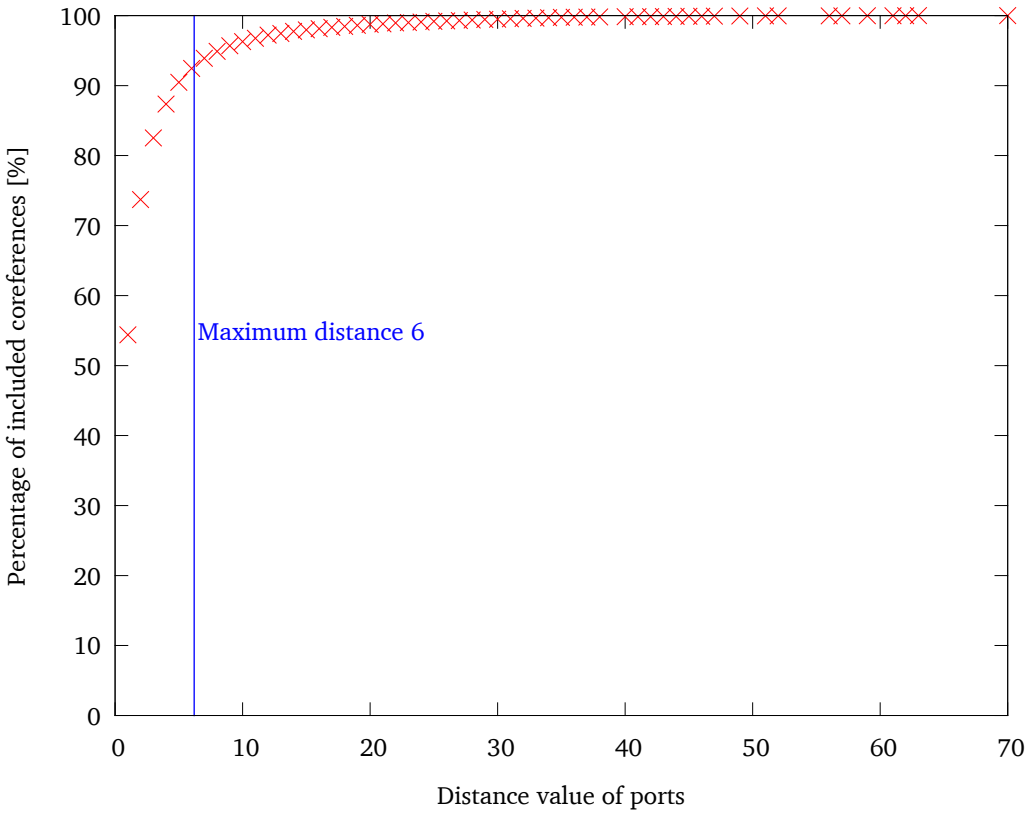

Figure 3.2.: Included coreferences with various maximum distances

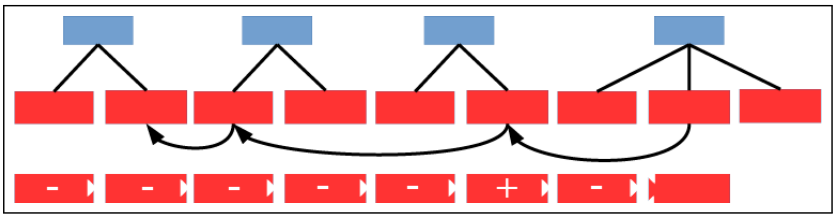

Figure 3.3.: Construction of positive and negative instances and coreference chains

Many existing contributions use other methods of creating training instances. Soon et al [11] also created positive instances for every noun phrase to its closest coreferent noun phrase, but negative instances for every pairing of the source expression to every other possibility in between this coference. With this method, the amount of negative instances is directly dependant on the distances of the coreferences. This can easily insert a bias into the distance value of the instances and omits negative instances on noun phrases without a coreference connection, whereas the proposed method created a limited distance window, but the distance value can be freely used as a feature in the learning process.

As a next step, the full set of instances was split in two parts: A training set that is used to train the various machine learning models, and a test set to put this model to a proof and analyse its performance. The percentage of training data versus test data is another degree of freedom in this implementation, and most experiments used a 50/50 distribution. It is important to notice that all instances with the same source port were allocated to the same set. In a test scenario, a decision has to be made for a single node to find its coreferent partner, or to determine that there is none. This scenario naturally implies that each of these decisions include all possible choices for the source port and all possible pairings to

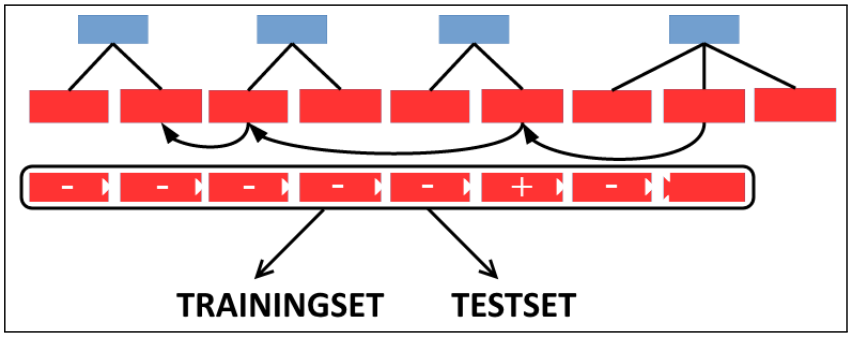

Figure 3.4.: Splitting the data: all related instances are put in the same data subset

other ports. Splitting these choices into two halves is not desired, as both training and testing needs all possible choices.

3.3 Experimental settings and Features

The Weka experiments were performed on the server of the Knowledge Engineering Group of TU Darmstadt. As only one system was used for all experiments, runtime and other performance measures can be compared without problems, and undesired variation from hardware or software differences was eliminated. The effect of various distributions between test and training data was examined, but all following experiments whose predictions were compared with performance measures were conducted with exactly the same test and training set. The whole data set consisted of 3145 documents in GraphML format. These documents were converted into 1,366,814 data instances by transforming the GraphML files into an ARFF file. They were split into a training set and a test set each containing 50% of the instances.

Table 3.1 contains a list of the used feature set for every instance. Every instance contains the unique identifier of the source node. With this identifier, the instances can be sorted by source node, and the decisions for each source node can be separated for evaluation and ranking purposes. Most of the node and port attributes of the GraphML files were directly used as features. Additionally, the computed distance value of the two verb nodes and seven identity features were added. The machine learning algorithms are not able to determine these identities themselves, but they are suspected to be an important indicator.

Although only about 30 attributes were used, some of them had over 2000 different possible String values. This lead to massive runtime and memory problems when running Weka experiments. It is much easier for these algorithms to handle boolean attributes. For this reason, every String attribute is transformed into several boolean attributes, so every possible value of the attribute is represented by a single boolean expression. As an example, a String attribute ”color” with three possible values ”red”, ”blue” and ”green” is split into three boolean attributes ”color=red”, ”color=blue” and ”color=green”. Exactly one of these three attributes is true for every instance in the database, given that the original attribute ”color” is always filled and never empty.

Numerical attributes could have been processed in a similar way, but they were left unchanged so the learning algorithm can still apply comparisons operators like $>$ or $<$ in the learning process. Boolean attributes like the class attribute were carried over into the final feature set without any changes. In its original state, 30 attributes were used, 26 of them are String format, three numerical and one boolean class attribute. The converted feature set had about 7000 attributes, three of them are numerical, the rest boolean. Since there were only 30 attributes in the first place, only 30 of the 7000 attributes of the

Feature name	Description	Format	Amount
Source	Unique identifier of source node	String	1
SrcLemma	Source node: Infinitive verb form	Boolean	3346
SrcPos	Source node: Part-of-Speech tag	Boolean	6
SrcStanfLabel	Source node: Stanford Label	Boolean	126
SrcVoice	Source node: Active / Passive	Boolean	2
SrcBindRahmen	Source node: Combined gram. functions of ports	Boolean	7
SrcPortGramFct	Source port: "S"ubject, "D"irect object, "I"ndirect object	Boolean	3
SrcPortPos	Source port: Part-of-Speech tag	Boolean	35
SrcPortNer	Source port: Named-entity recognition	Boolean	13
TarLemma	Target node: Infinitive verb form	Boolean	3348
TarPos	Target node: Part-of-Speech tag	Boolean	6
TarStanfLabel	Target node: Stanford Label	Boolean	128
TarVoice	Target node: Active / Passive	Boolean	2
TarBindRahmen	Target node: Combined gram. functions of ports	Boolean	7
TarPortGramFct	Target port: "S"ubject, "D"irect object, "I"ndirect object	Boolean	3
TarPortPos	Target port: Part-of-Speech tag	Boolean	35
TarPortNer	Target port: Named-entity recognition	Boolean	13
Distance	Computed verb distance	Numeric	1
LemmaGleich	Identity of source and target verb infinitive forms	Boolean	1
TokenGleich	Identity of source and target Verb tokens	Boolean	1
PortLemmaGleich	Identity of source and target port argument heads	Boolean	1
PortTokenGleich	Identity of source and target port argument tokens	Boolean	1
BindRahmenGleich	Identity of source and target verbs port combinations	Boolean	1
StanfLblGleich	Identity of source and target verb Stanford Labels	Boolean	1
PortGrmFctGleich	Identity of source and target port gram. functions	Boolean	1
AnzahlScrVerbRels	Source verb: Amount of matching verb relationships	Numeric	1
AnzahlTarVerbRels	Target verb: Amount of matching verb relationships	Numeric	1
class	Real coreference label	Boolean	1

Table 3.1.: Feature set for Weka experiments

feature set can be non-zero. The used Sparse format solves this problem as each instance in the data section of this file only contains the values that are non-zero together with the corresponding attribute number, as shown in figure 3.5.

See section A.1 in the appendix for more information about the various program steps.

3.4 Coreference Resolution with Decision Trees

To apply the machine learning algorithms on the resulting data files, version 3.6 of the Weka toolkit from the Machine Learning Group at the University of Waikato was used. It is an open source software

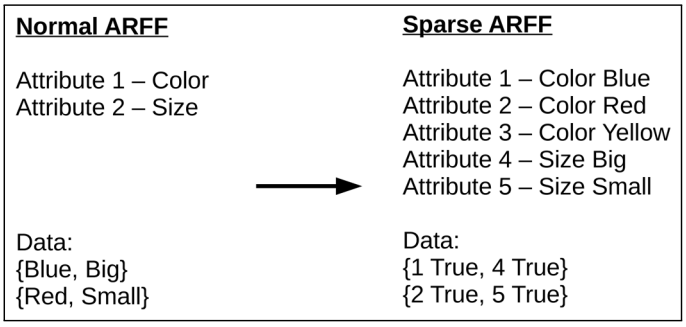

Figure 3.5.: Example sparse conversion

containing many different machine learning algorithms, filters and other helpful tools.
The first experiments used the J48 algorithm that is implemented in the Weka toolkit. As described in chapter 2.4.1, this is a decision tree algorithm that uses all available training data and attributes to create a single decision tree, which can be used to classify test instances by simply processing the new instance through the tree. The resulting leaf node contains the prediction for the class attribute of the instance. The implementation of this algorithm contains various parameters that mostly provide pruning options, the experiments used default settings.
The Random Forest implementation in Weka introduces two parameters that are easy to use and very important, and have a big impact on the results. The first parameter determines the number of separate decision trees that are constructed during the learning process. Increasing this parameter obviously increases the runtime of the learning process, but the effect on the performance had to be examined. The second parameter controls the amount of random feature attributes that are considered at each step of the construction of the decision tree. By increasing this parameter each of these construction steps takes more time as more features have to be tested and compared, but on the flip side, chances are getting higher that the best possible feature lies inside the random selection. Various experiments were conducted to test the effects of this parameter, and the combination of both parameters, too. The prediction outputs of the Random Forest algorithm has exactly the same format and content as the output of J48, so they were processed in the same way. Also, since the same training and test sets were used, the same performance measures could be applied and directly compared witch each other.

3.4.1 Ranking

The classification predictions in the previous chapters were made under the assumption that every instance the test set is completely separate from all other instances. This assumption made it easy to quickly compare different machine learning algorithms or different parameter settings, but this assumption did not match the assumed field of application. When searching for the nearest coreferent expressions within a given document, it is unnecessary to make predictions about every possible pairing of expressions in the considered vicinity, only the most likely combination has to be determined. In this case, the scenario changes from predicting every pairing to ranking all possible pairings of every single port. Given a single port, all valid pairings that contain this port as the source port form its selection of possible pairings. If a coreferent connection has to be determined for this port, the pairing with the highest probability of coreference has to be returned as the solution, as shown in an example in figure 3.6. This is a closer resemblance to the present data, as the given database also only contains a single outgoing coreferent connection from each verb argument.

With this motivation, the classification results of the J48 and Random Forest experiments were transformed to ranking results. All instances with the same source port were gathered to a selection, and

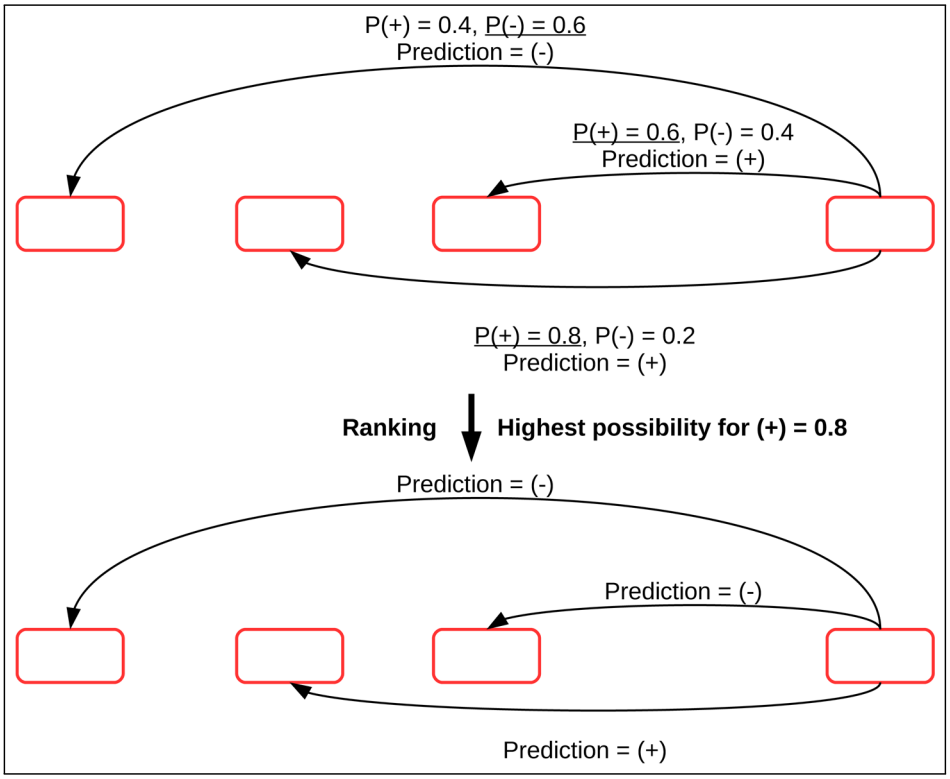

Figure 3.6.: Example for ranking reduction to one positive prediction

within these selections, only the instance with the highest predicted probability to have a positive class attribute was now predicted to be positive, all other instances were predicted to be negative. This could cause instances that were formerly predicted as a positive instance to be changed into a negative prediction if there is another possible instance with the same source port but higher probability to belong to the positive class. On the other hand, many instances would be changed from a negative prediction to a positive prediction, even if the calculated probability of them being positive is under 50%. This is possible if every other instance in its selection has even lower probability values for a positive prediction, thereby creating very high numbers of false positive errors by assigning a coreference connection to every single argument all documents. To solve this issue, the ranking problem was split into two separate tasks. In a first step, every argument is examined whether a coreference should be determined or not. If a coreference has to be found, the most possible pairing for this coreference in searched as a second step. This second step exactly matches the explained transformation from the classification results into ranking results. The first step, to determine whether a coreference has to be found or not, was not pursued in this work, and an omniscient oracle was assumed and simulated. Given the gold annotated data set, this oracle returns ”true” for every port that has an outgoing coreferent connection in the GraphML file, otherwise it returns ”false”. In the case of ”true”, the predictions of all instances with this source port are transformed into ranking results in the explained manner. Otherwise, all corresponding instances are predicted to be negative, as they will not be ranked at all.

The transformation to ranking results offer additional opportunities for experimentation by introducing a minimum probability threshold. If the probability for a positive class attribute for all instances in a selection is below this threshold, none of them is predicted to be positive. This introduces a flexible option to limit the positive prediction of the ranker in hopes of improving the remaining positive prediction, and thereby the Precision value of the ranker. By setting a very high threshold of 0.9 or higher, as shown in figure 3.8, the ranker may be utilized to give only very few predictions, but hopefully with a

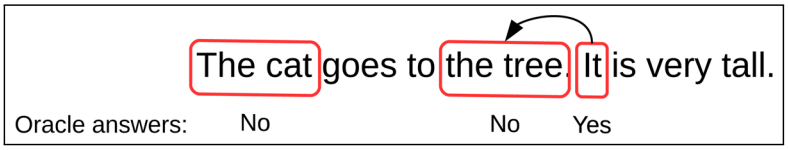

Figure 3.7.: Oracle answers

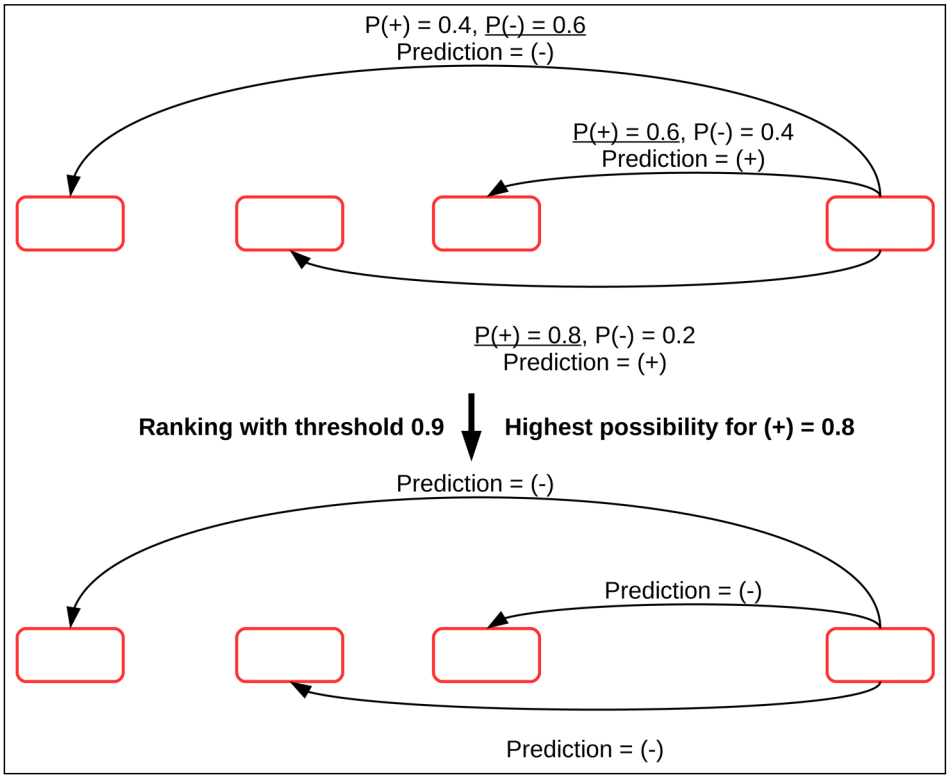

Figure 3.8.: Example for ranking with a high threshold

very high true positive ratio.

3.5 Impact of Features

The conducted Weka experiments did not only yield prediction results, but also the constructed decision trees. These learned models were analysed to determine the most important attributes of the data set that were used as features in the learning process. The trees were traversed while adding up scores for every feature that was used in a decision node. These scores correlate with the level where the feature was found. If the feature was used in a root node as a first split of the instances, the value 1 was added to the features score. In the second level of the tree, a feature would only score 0.5, but they can potentially be used in the other branch as well, thus adding its score to 1 again. Each additional level divides the score in half to reflect the diminishing relative importance. In this manner, many different trees of different Random Forest experiments were accumulated to a single score list.

3.6 Combination with Existing System

The machine learning approaches in this work were never designed to conquer the problem of coreference resolution as a stand alone solution. The real application scenario was meant to be an integration into existing coreference resolution software as an added resource. As a simulation of this approach, the ranking experiments were combined with the Stanford Natural Language Processing tool set ”Stanford CoreNLP” within the applied data model. For this approach, the whole gold annotated data set was also annotated using Stanford CoreNLP. This data could be compared to the real gold annotations, and then compared again when adding parts of the ranking results that were gathered during the experiments.

As mentioned, the gold annotated data was transformed into Weka ARFF format with all defined restrictions and assumptions explained in chapter **??**, like maximum distance between verb arguments. To compare the Stanford CoreNLP annotations with the gold annotations, they had to be matched to the same experiment environment, and were transformed into prediction outputs in the same format as a Weka output file. For every instance in the test set, the Stanford CoreNLP annotated data was searched for an exact match. If this data contained a coreferent connection that matched the features of the instance in the test set, this instance was predicted as positive. Otherwise, it was predicted as a negative instance. For the matching of an instance in the test set, the Stanford CoreNLP data is searched for the same document, and exact matches in verb lemma of both source and target node, and grammatical function of these nodes. There is no direct identification of the involved ports because even the identification of verbs and ports is part of the annotation process. One verb in the gold annotated data may not even exist in the annotations from Stanford CoreNLP, so every match has to be cross searched.

After matching the annotations, the usual performance measures of the Stanford CoreNLP annotations can be examined, considering the explicit data model. These performance scores can now be observed while combining the annotations with various ranking results. For this combination, the ranking is added as an additional chance to give positive predictions, hopefully adding true positive hits while avoiding new false positive errors. Every positive prediction of CoreNLP is left untouched. For every negative prediction, if the ranker would have predicted the instance to be positive, then the prediction of CoreNLP is changed to positive. The Precision values before this procedure and afterwards were compared to investigate the effect of this approach. Because only positive predictions are added, the recall value of the entire setting can only increase, making this evaluation measure irrelevant here.

3.7 Motive Based Approach

The given data structure in GraphML format offered possibilities for additional analysis on the graph structure. The core idea was to find motive signatures in the nodes and edges of the graphs, and study their different shapes and appearances to gain information about often reoccurring structures. The most simple motive was defined to be two verbs in the specified maximum distance, which can either have a coreference connection between two of their ports or not. In this case, the verbs are defined by their verb lemma and their argument frame, so the verb ”go” with only a subject argument is counted separately from the verb ”go” with both subject and direct object arguments. The 2-way motive has been searched and counted for the whole gold annotated dataset. To gain additional information of common motives, this counting was also performed on the much bigger Gigawords corpus that contains a very large set of automatic annotated documents. Bigger 3-way motives contain three verbs in different connection combinations. Please note that there are three verb nodes involved, not three ports. As each port in the given data structure can only have a maximum of one incoming edge and one outgoing edge, there is only one possible motive containing three connected ports. One verb node can have up to three different ports, thus allowing incoming and outgoing edges of the same amount. There are four different connec-

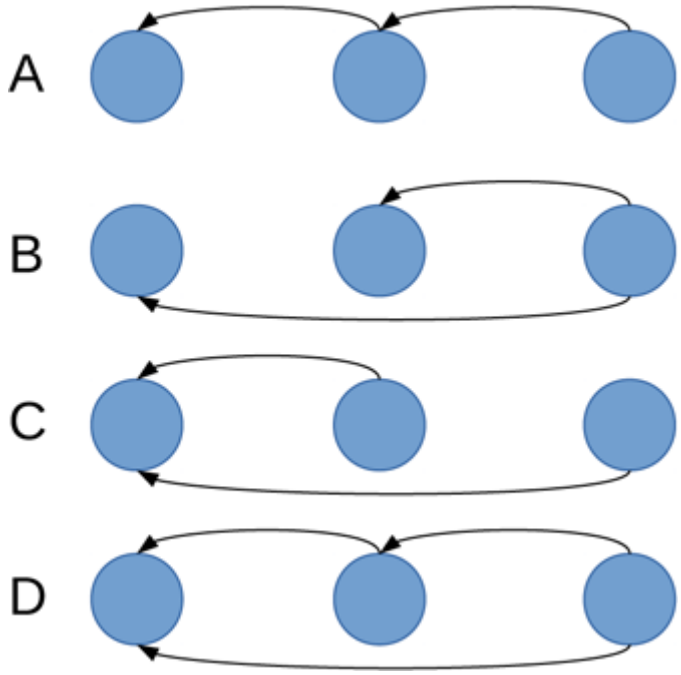

Figure 3.9.: Four possible combinations involving three verbs

tion combinations involving three verbs, as shown in figure 3.9. These motives were also searched and counted for future analysis.

As a first application of this data, a simple classification was performed that utilizes the 2-way motive appearances. The counting was performed on a training set. For each pair of verbs, its probability for a coreference connection was calculated by dividing the amount of found connections between the separated verbs by the total amount of occurrences. Each instance in the test set could then be assigned the corresponding probability value from the motive counting. In case this combination was not found in the training set, a general connection probability was picked. This general connection probability was computed by counting any connection of the verb in relation to its overall occurrences, regardless of any pairings. The assigned probabilities were turned into ranking results with the same procedure that was used to transform the Weka probabilities into ranking results. These ranking results were now analysed with the usual performance measures.

4 Evaluation

This chapter contains results and conclusions of the implementation described in the previous chapter, following each of the formulated research question. Some raw results are presented and pictured in both tabular and graphic form, followed by a brief explanation of their relevance. Precision, Recall and F-Measure are used as performance measures, as described in the basics chapter 2.4.3.

Precision	$\frac{TruePositives}{FalsePositives}$
Recall	$\frac{TruePositives}{FalseNegatives}$
F-Score	$2 * \frac{precision * recall}{precision + recall}$

Table 4.1.: Evaluation measures

4.1 Coreference Resolution with Decision Trees

- Can coreferences be predicted using lexical features?
- Which approach appears to be more / less successful?

To cover these first two research questions, the results of the machine learning approaches were examined and compared, and the advantages and disadvantages of different algorithms and settings contrasted.

4.1.1 J48 Results

The huge problem of the J48 experiments was the very long runtime. Without additional data modifications, a single run of J48 on the 683407 instances in the training set needed about three days and six hours of computation on the server. Precision values averaging at 0.505 were achieved, with a Recall of 0.148, resulting in an F-Score of 0.229. These values had to be improved and optimised, but since every new run needed additional three days, the main focus had be improving the runtime. The big imbalance between positive and negative data offered an obvious chance to improve the runtime by downsampling the negative instances. Random downsampling with various rates provided adequate speed enhancements, but the Precision value went down tremendously. By randomly leaving every fifth negative instance in the training data set, significantly more positive predictions were made by the trained model, but only with a Precision of 0.287. The much higher amount of positive predictions resulted in a much better Recall of 0.493, combining to an F-Score of 0.362. The F-Score could be improved, but the much lower Precision was very much undesired, and the runtime was still over twelve hours. Higher downsampling rates of 1/10 or 1/100 cut down the runtime to four hours or thirty minutes, respectively, but with progressively worse performance measures in every aspect, as seen in table 4.1 and figure 4.1.

Since the parallel tests with the Random Forest algorithm featured much better results, with better performance values and significant runtime improvements, additional tests with J48 were cancelled to focus on the Random Forest experiments.

% negative data	Precision	Recall	F-Score	Minutes
100	0,5050576	0,14841266	0,22941194	4680
50	0,39671325	0,29684623	0,3395897138	1954
20	0,28681037	0,49382818	0,3628697	683
10	0,225723	0,6663502	0,33721575	244
1	0,08444339	0,95260704	0,15513492	26

Table 4.2.: J48 Results - 50/50 training/test set distribution, training set with random sampled negative instances, standard feature set, J48 algorithm with default settings

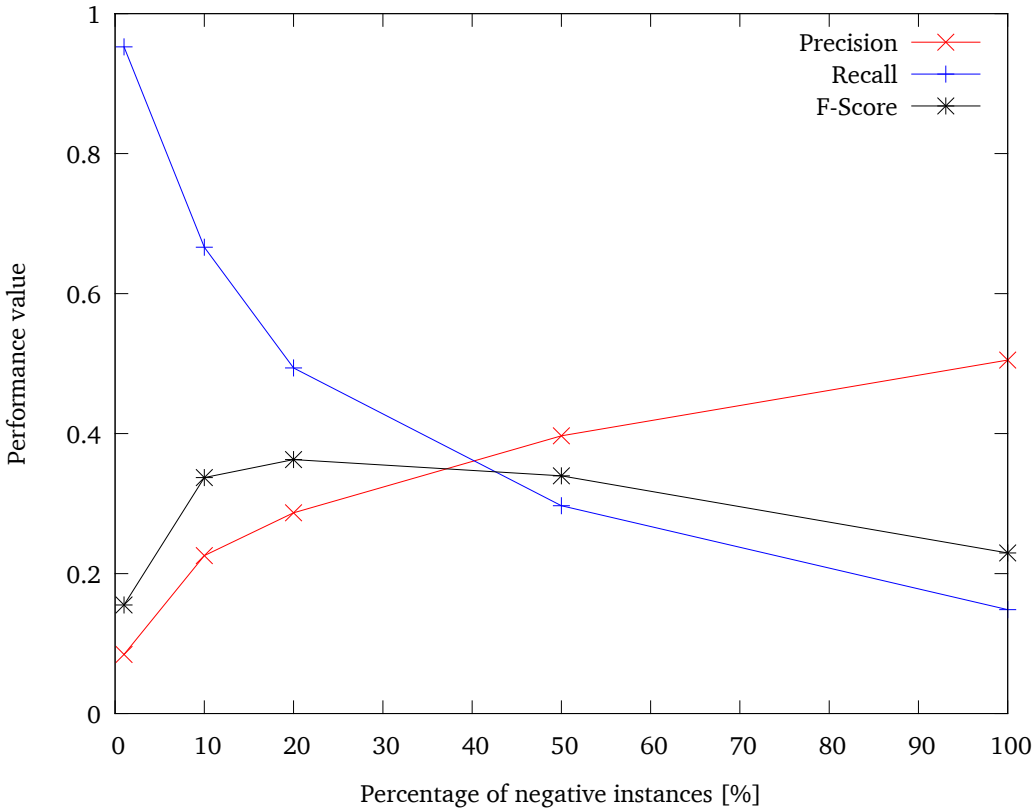

Figure 4.1.: J48 results

4.1.2 Random Forest Results

The first run of Random Forest was started utilizing the default configuration of the two explained parameters, amount of constructed decision trees and amount of randomly drawn features per construction step. In this configuration, ten decision trees are constructed. In each construction step, the number of considered features is based on the base two logarithm of the amount of features, which results to thirteen in the case of the used Sparse ARFF file. With these basic settings, Weka needed 29 minutes to build the model and test it on the test dataset, which is a huge runtime improvement over J48. The Recall value was only 0.11, which is a bit worse than J48 with its full run, but the Precision value of 0.660 is much better. These results were very similar to the results of the J48 experiment, but could be computed in a fraction of the time. Various parameter settings were tested with the aim of improving the performance values while keeping the runtime reasonably short and manageable.

Increasing the amount of decision trees without also adding more feature attributes to each construction step had only a very small impact on the performance. With 100 trees, the Precision raised to 0.702 and Recall to 0.128, but adding more trees did not result in any measurable improvements, as seen in table 4.3 and figure 4.2. The same values for Precision and Recall were observed even with 500 decision trees, while the runtime of the training process grew very linear to the amount of trees. Constructing 100 trees took about four hours 29 minutes, and the experiment with 500 trees finished after nearly 24 hours.

Trees	Precision	Recall	F-Score
100	0,70258623	0,12757424	0,2159387433
200	0,7024548	0,12737857	0,2156521807
300	0,7051529	0,12517732	0,2126121586
400	0,7038781	0,12429682	0,2112834074
500	0,7049407	0,124932736	0,2122496433

Table 4.3.: Random Forest results with varying trees - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with varying amounts of trees

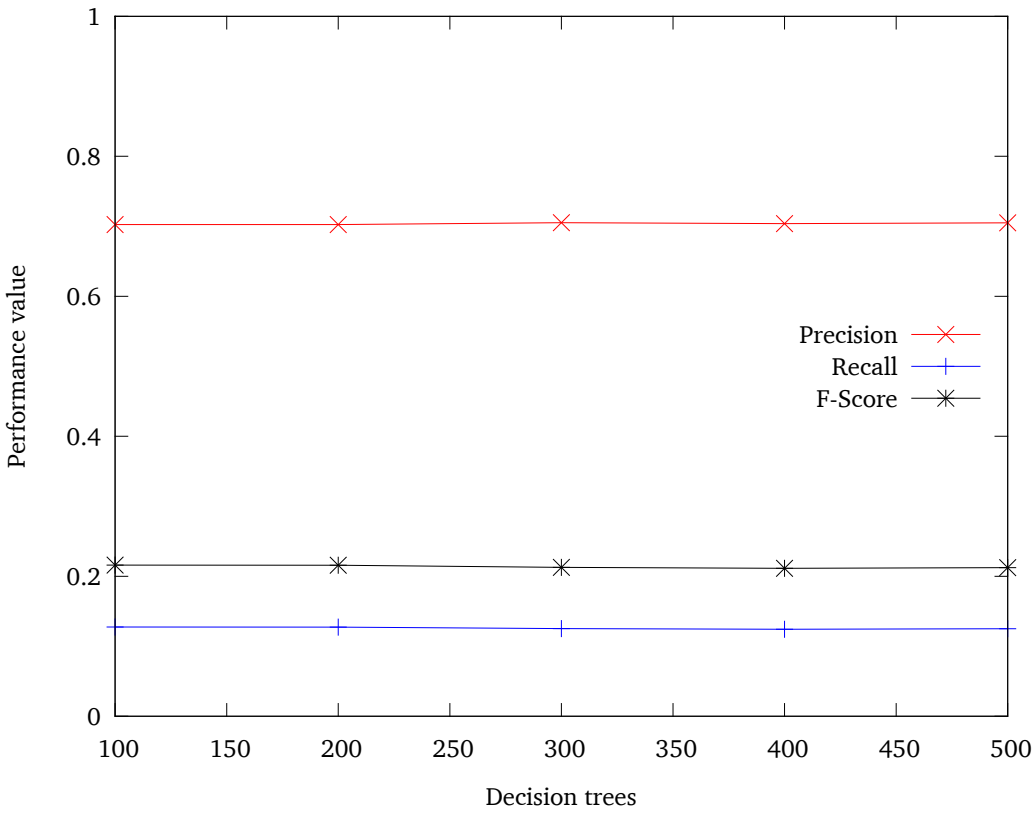

Figure 4.2.: Random Forest results with varying trees

Increasing the amount of features per construction step had less than linear effect on the runtime. Setting this parameter from default settings to 100 raised the runtime from 29 minutes to 41 minutes average. With 1000 features per draw, about two hours 50 minutes were needed for the training process, instead of six hours and 40 minutes that would be expected if the relation between amount of features and runtime was linear.

Increasing the parameter up to 1000 features had positive effects on all performance measures, as seen in table 4.4 and figure 4.3. Precision values got up to 0.700, with a Recall of 0.334, which is much better than any previous experiment with Random Forest, as was the resulting F-Score of 0.450.

Features	Precision	Recall	F-Score
100	0,68028057	0,2134716	0,3249683449
200	0,69614995	0,270655	0,3897714109
300	0,70423025	0,2956024	0,416413991
400	0,70496285	0,3154625	0,435875771
500	0,70283073	0,31213617	0,4322877765
600	0,699647	0,31996283	0,4391111728
700	0,7012418	0,32871887	0,4476120668
800	0,6949715	0,3278873	0,4455596974
900	0,6974886	0,3287678	0,446889866
1000	0,69002426	0,33429536	0,4503904912

Table 4.4.: Random Forest results with varying features - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with varying amounts of features

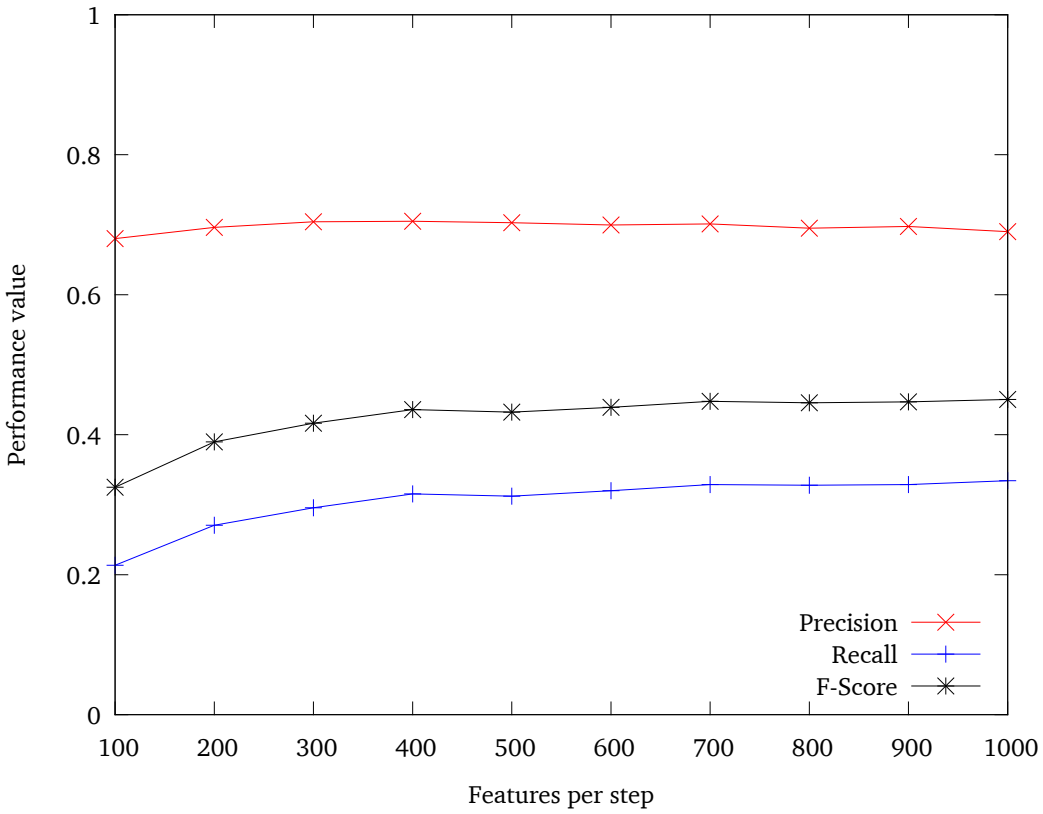

Figure 4.3.: Random Forest results with varying features

Combining more decision trees with higher amount of features per step was the obvious next step. As seen in the previous experiments, adding more trees has a big impact on runtime, but very diminishing

returns in performance increases after reaching about 100 trees. In this manner, combining 100 trees with 100 features per step did good results with a Precision of 0.731, Recall of 0.247 and F-Score of 0.369, but increasing the amount of trees ten times to 1000 did not have any measurably different result. Increasing the amount of features and trees together had small positive effects, as shown in table 4.5 and figure 4.4. On the flip side, increasing the amount of features ten times to 1000 had a big large impact on the overall performance, with a Precision value of 0.726, Recall of 0.340 and F-Score of 0.468. This was the best score that was achieved using the described methods, adding more trees had again no measurably improving effects. This experiment with 100 trees and 1000 features took 28 hours to compute. Since roughly one day of computing was considered to be the upper limit of manageable computation time, no further experiments with even higher parameters were planned.

Features_Trees	Precision	Recall	F-Score
100	0,73188406	0,24702832	0,3693815575
200	0,7297905	0,30161914	0,4268309592
300	0,7266314	0,32245755	0,4466881116

Table 4.5.: Random Forest results with varying trees and features - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with varying amounts of trees and features

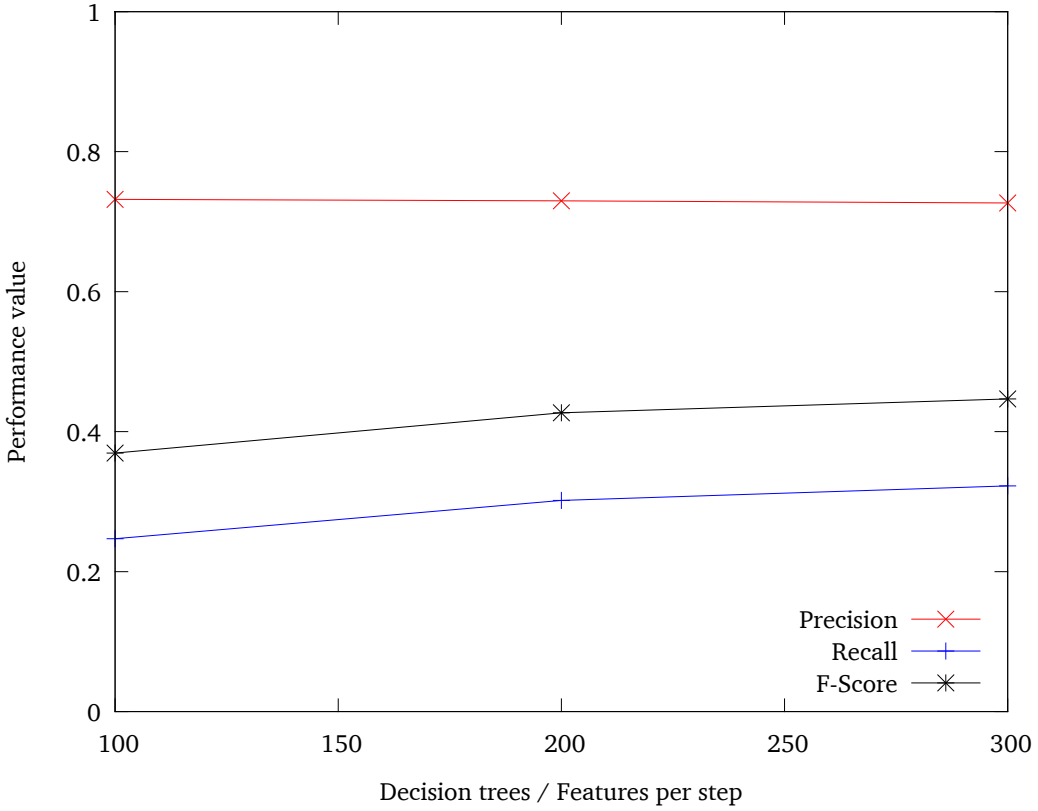

Figure 4.4.: Random Forest results with varying trees and features

4.1.3 Classification Conclusions

The used training set with 683407 and 7000 attributes provided a big challenge for the J48 algorithm. Since a split with every remaining attribute has to be tested for all relevant instances at every deci-

sion node, the computation time gets very unwieldy and unmanageable for tuning experiments. The algorithm uses no random elements in its decision tree construction, and uses all attributes, so it was expected that the results should be very good, even at default settings. Instead, the Random Forest experiments provided much better performance values, running at default settings, too, but in much less time. This hints to overfitting problems of the J48 algorithm that could be improved by limiting the maximum tree depth or increasing the pruning parameters, but time and performance values of Random Forest made these possibilities unattractive for the current work. Random Forest handles the large amounts of data and features very well with its random draws, but the results were surprisingly good. Tuning via the two parameters - amount of trees and features per construction step - turned out to be a very flexible trade-off between performance and runtime as each increase in these parameters caused results of better or at least equally good quality. Random Forest proofs to be a very potent classification algorithm with this big data, even when using simple features.

Just like all other results of this work, the specific performance values have to be regarded with care and should not be heedlessly compared to other coreference resolution systems. The applied data model, especially the limiting maximum distance between verb arguments, form a very different application scenario. Comparing and improving the performance in this special environment should nevertheless give strong hints about algorithms and parameter settings that should have good results for the given data, even in another scenario.

4.1.4 Ranking results

The obtained ranking results were analysed with the same performance measures as the pure classification results. By using an oracle for determining if a coreference connection has to be found or not, the ranker is forced to a decision at every port that has such a connection in the gold annotated data, at least without utilizing a threshold. Because of this, the amount of positive predictions from the ranker always exactly matches the amount of coreference annotations. This eliminated the examination of how many positive prediction were made, leaving the precision of these predictions as the only interesting aspect. Every false positive error directly implies that the real coreference has been missed, resulting in an additional false negative error. Therefore, Precision and Recall correlate, and the results concentrate on the Precision value of the different experiments. It is important that the Recall scores of classification experiments may not be directly compared with Recall scores of rankings, because the ranker uses an oracle that gives him perfect information about all source nodes that have a coreference connection.

Without using a minimum probability threshold for the ranker, the Precision value of a classification result was higher than the value of its corresponding ranking result. This is shown in table 4.6 and corresponding figure 4.5, and stands true for every conducted experiment, but the difference varies and correlated strongly with the amount of decision trees of the Random Forest algorithm. Leaving this parameter at its default setting of 10 results in a ranker with a Precision value of 0.578 average, which is about 0.099 lower than the corresponding prediction Precision. The lowest difference was achieved by using high settings for both parameters. The lowest difference of 0.032 was attained by a Random Forest run with 300 trees and 1000 features per construction step.

A threshold introduces a trade-off between the amount of predictions and Precision, as shown in the results in table 4.7 and figure 4.6 of an experiment with 100 trees and 100 features, but varying threshold settings. All ranking Precisions at threshold 0.5 or higher were significantly higher than the Precision of their corresponding classification results, although the amount of positive predictions went down at least to one third of the amount without a threshold. At standard settings, a Precision of 0.817 was reached, while the threshold filtered out about 75% of the positive predictions. The best results were again achieved with 300 trees and 1000 features per step, resulting in a Precision of 0.892 with the

Trees	Precision	Recall	F-Score	Prec.(Rank)	Rec.(Rank)	F-S.(Rank)
100	0,70258623	0,12757424	0,2159387433	0,63148004	0,63148004	0,63148004
200	0,7024548	0,12737857	0,2156521807	0,64270425	0,64270425	0,64270425
300	0,7051529	0,12517732	0,2126121586	0,6472461	0,6472461	0,6472461
400	0,7038781	0,12429682	0,2112834074	0,6487079	0,6487079	0,6487079
500	0,7049407	0,124932736	0,2122496433	0,6505351	0,6505351	0,6505351

Table 4.6.: Precision comparison of ranker without threshold and classifier - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with varying amounts of trees, with and without ranking

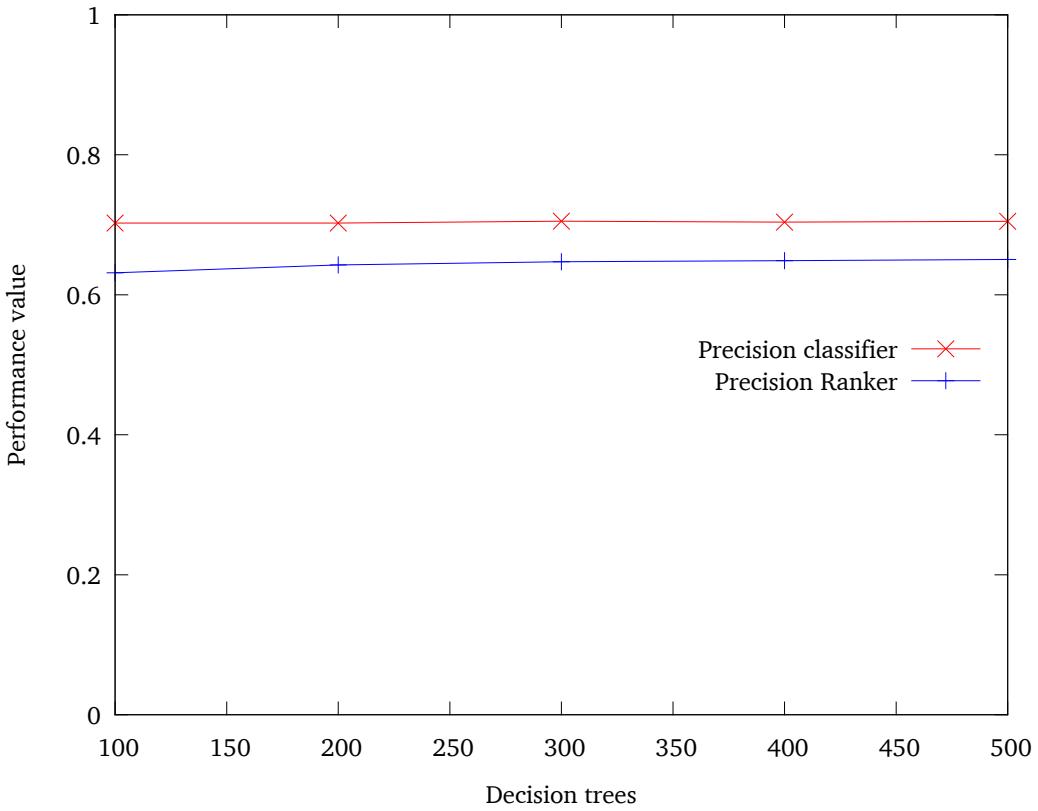

Figure 4.5.: Precision comparison of ranker without threshold and classifier

threshold filtering out nearly 65% of the positive predictions. Raising the threshold even higher enhanced all effects. At a threshold of 0.7, the ranker with 300 trees and 1000 features per step reached a Precision of 0.933, but only 18% of the positive predictions were left by this threshold, blocking out the other 82%.

4.1.5 Ranking Conclusions

The ranking results were implemented to better reflect the real application scenario. In the original classification problem, multiple coreferencial connections could be found for any single argument. This is possible in a real world scenario if all these connections belonged to the same coreference chain and all involved arguments referred to the same entity. In the experiment environment, only one specific connection has to be found, only the nearest connection. The classification predictions allowed for many possible positive results, but multiple nearest connections are not possible and pointless. Reducing the

Threshold	Precision	Recall	F-Score	% positive predictions
0	0,66049534	0,66049534	0,66049534	100
0,1	0,7146731	0,6050510232	0,6553092162	84,66122808
0,2	0,7762349	0,4997313149	0,6080237591	64,37887744
0,3	0,82639575	0,4092618882	0,5474207007	49,52371648
0,4	0,8651074	0,3107810961	0,4572865997	35,92399003
0,5	0,89014375	0,2117629849	0,3421333067	23,78975136
0,6	0,89848065	0,1271066368	0,2227072334	14,14684187
0,7	0,91166323	0,0645303122	0,120529199	7,0783059
0,8	0,90016365	0,0268672752	0,0521772119	2,98471008
0,9	0,91943127	0,0094768207	0,0187602744	1,03072639

Table 4.7.: Precision and amount of positive predictions of rankers with different thresholds - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with 100 trees and 100 features, with ranking and varying thresholds

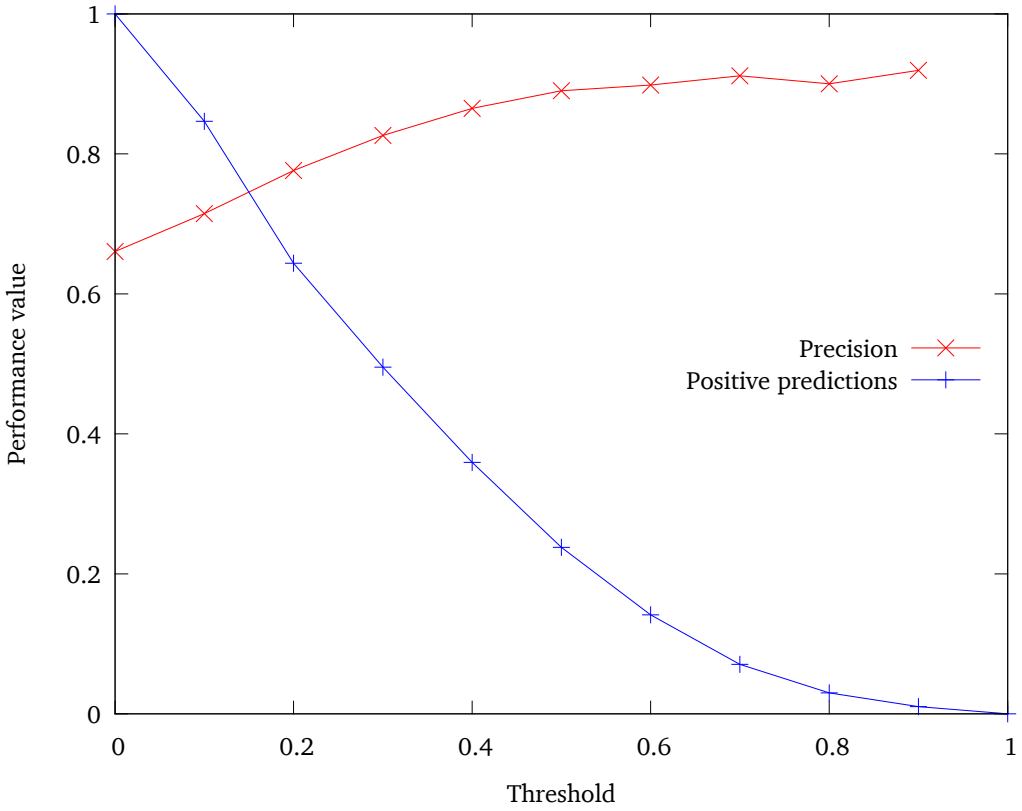

Figure 4.6.: Precision and amount of positive predictions of rankers with different thresholds

amount of positive predictions per argument to at most one was the logical consequence.

Introducing a minimum probability threshold for the ranker limited the amount of positive predictions with the goal of improving the Precision of the remaining predictions. A threshold of 0.5 eliminated all predictions that were classified as negative instances in the first place, because the predictions for a positive class attribute was below 50%. The positive predictions were still limited to the ports that really

have a coreference in the gold data, and narrowed down to the most likely instance in case multiple instanced were predicted to be positive. With this threshold setting, the Precision value of the rankers had to almost always surpass or at least match the classification Precision in all experiments. Eliminating all predictions of ports that have no coreference by using the oracle can only eliminate false positive errors, and this could never result in a higher precision. Ranking all instances with the same source port can also rarely introduce more errors than hits, given a minimum threshold of at least 0.5. Different cases for every selection with the same source port have to be considered, regarding the original classification results:
1) All instances of the selection were predicted to be negative. The ranker would normally select the instance with the highest probability for a positive result. In this case, this probability has to be lower than 0.5 because all instances were predicted to be negative. Without a threshold, this could introduce an additional false positive error if the wrong instance was chosen. The threshold of 0.5 eliminates this decision, as no instance will be predicted as positive.
2) Exactly one instance of the selection was predicted to be positive. The ranker exactly would duplicate the decision of the classification, as this particular instance has to be the instance with the highest probability to be positive, and the probability has to be higher than the threshold of 0.5.
3) More than one instance of the selection was predicted to be positive, none of them was the real positive instance. In this case, all the positive predictions are false positive errors. The ranker predicts only one of these instances as positive, thereby reducing the amount of false positive errors to one.
4) More than one instance of the selection was predicted to be positive, one of them was the real positive instance. If this true positive instance had the highest predicted probability to be positive, the ranker selects it, keeping the true positive prediction and eliminating all false positives. If the true positive instance had not the highest predicted probability, then this can be the only case where the ranker leaves the false positive prediction in its results, but the true positive prediction is eliminated, together with potential additionally false positive predictions.

The raw Precision values have to be taken with care, as mentioned several times before, but when using an adequate threshold, the much higher Precision values compared to the prediction values are still a strong hint that this procedure in a step in the right direction when Precision is important. A ranker of this sort may not be suitable to find all coreferences in a document, but may be used to find at least some coreferences with a very high success rate. An application for this could be a first scanning of a new document to find just a few coreferences as a first base line. These coreferences may be extended manually or with the help of another system. Another application scenario could turn this idea around by using the presented ranker after another system has already proposed results for coreferencial connections. The ranker may verify the results to find unlikely predictions, or give suggestions for additional coreferences that the previous system has missed. An approach for this last idea will be presented in the following chapter.

4.2 Impact of Features

An extract of the best scores can be seen in figure 4.8. The top 20 features list contains five identity features that contain the value 1 if a specific attribute value of the source port matches with the value of the same attribute at the target port. These features had to be inserted into the feature pool as the used machine learning algorithms only include single features for each test and can never detect equalities. Apparently, these features are a very good addition in the feature set. The distance feature also has a very high position in the scoring, but this position may be misleading since the distance feature is numeric. It ranges from one to the set maximum distance parameter, which was six for the conducted experiments. Since splits with numeric features are made with ”<=” and ”>” tests, this feature may appear several times even inside the same branch of the decision tree, whereas boolean features can only appear and be tested once for every branch. Nevertheless the very high second position of the distance feature may still

Score	Feature	Description
14.059	SrcPortPos_005	Source port: Part-of-speech tag = Determiner
14.015	Distance	Computed verb distance
11.820	AnzahlSrcVerbRels	Source verb: Amount of matching verb relationships
10.990	PortLemmaGleich	Identity of source and target port argument heads
10.502	TarPortPos_004	Target port: Part-of-speech tag = Proper noun, singular
9.620	SrcPortPos_000	Source port: Part-of-speech tag = Noun, singular or mass
8.985	LemmaGleich	Identity of source and target verb infinitive forms
8.790	TarPortGramFct_000	Target port: Grammatical function = Direct object
8.398	SrcPortGramFct_001	Source port: Grammatical function = Direct object
8.304	PortTokenGleich	Identity of source and target port argument tokens
8.139	AnzahlTarVerbRels	Target verb: Amount of matching verb relationships
7.879	TarPortGramFct_001	Target port: Grammatical function = Subject
7.349	PortGrmFctGleich	Identity of source and target port gram. functions
6.951	SrcPos_002	Source verb: Part-of-speech tag = Verb, 3rd person singular present
6.846	TarPortPos_008	Target port: Part-of-speech tag = Adjective
6.622	StanfLblGleich	Identity of source and target verb Stanford label
6.556	TarLemma_1279	Target verb: Infinitive verb form = regain
6.547	TarBindRahmen_001	Target verb: Only Direct object port
6.437	SrcStanfLabel_096	Source verb: Stanford label = prep_near
6.397	SrcPortGramFct_000	Source port: Grammatical function = Subject

Table 4.8.: Decision Tree Scores, top 20 results

point at its high relevance. Other specific features with very high importance scores could be compared to other similar experiments, or investigated specifically under natural language aspects.

4.3 Combination with Existing System

Adding a ranker without any thresholds resulted in more errors than additional hits. Using 100 trees and 100 features per step for a Random Forest algorithm and converting the predictions to a ranking without threshold added 5607 more true positives, but 6267 more false negative predictions. The Precision of CoreNLP in this scenario without additions was 0.623, and this got reduced to 0.559 with the additional predictions from the ranker. Adding a threshold had a big impact on these scores, maxing at a threshold of 0.4 with a slightly improved precision of 0.634, as shown in table 4.9 and figure 4.7. Better results at internal comparisons as shown in the previous sections lead to better results in this combination scenario.

4.3.1 Conclusions

The Random Forest experiments lead to ranking results that improved the Stanford CoreNLP Precision within this specific experiment environment. These improvements were achieved with relatively simple methods, using features that are obtained easily and an algorithm that runs quite fast. Combining a simple machine learning approach with a system with completely different focus, like CoreNLP, seems very

Threshold of Ranker	Precision without Ranker	Precision with Ranker
0	0,62342477	0,55943984
0,1	0,62342477	0,5886301
0,2	0,62342477	0,6139767
0,3	0,62342477	0,62952745
0,4	0,62342477	0,63433945
0,5	0,62342477	0,63267726
0,6	0,62342477	0,6291469
0,7	0,62342477	0,62635106
0,8	0,62342477	0,624212
0,9	0,62342477	0,6239064
1	0,62342477	0,62342477

Table 4.9.: Comparison of Stanford results with or without added ranker predictions - 50/50 training/test set distribution, standard feature set, Random Forest algorithm with 100 trees and 100 features, with ranking and varying thresholds, added after negative Stanford predictions

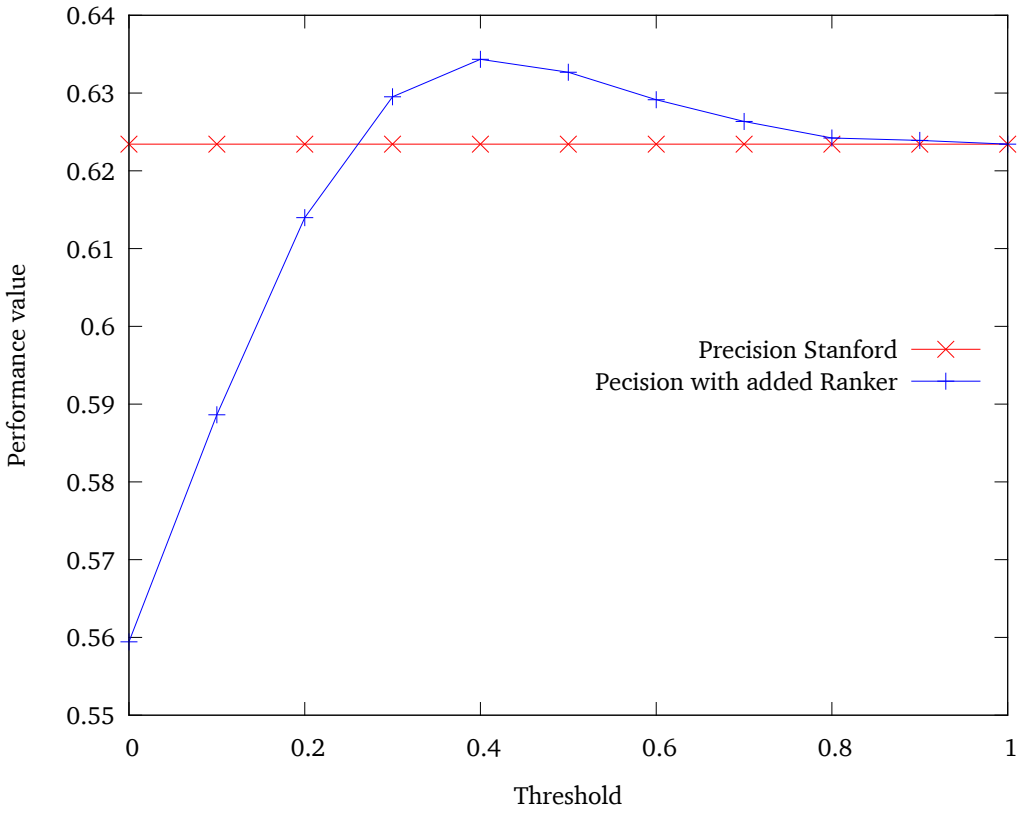

Figure 4.7.: Comparison of Stanford results with or without added ranker predictions

promising, although the presented results can not be simply carried over to a real application environment. Instead of transferring Stanford CoreNLP or its results into this environment, the Random Forest ranker has to be directly connected to the CoreNLP system.

CoreNLP uses a sieve structure where many different tests and decisions are executed consecutively. At each sieve level there may be a decision that a coreference was found, or that there is no coreference. Alternatively, no decision is made, and the problem is passed to the next sieve level. At any sieve level, the ranker can be inserted to give a prediction. Inserting the ranker at different sieve levels with different minimum probability thresholds would give an estimate about the effect of this combination in a real application environment.

4.4 Motive Based Approach

Without an additional minimum probability threshold, a Precision value of 0.337 was reached by this simple motive method. This values was increased to an average of 0.543 when using a minimum probability threshold of 0.5, although about 81.5% of the positive predictions were filtered out by the threshold, as shown in table 4.10 and figure 4.8. This motive ranker was also added to the Stanford CoreNLP as an additional chance for positive predictions in the same way as described in chapter 3.6. In these experiments, the motive ranker always added more errors than additional true positive predictions, even when using high thresholds. With the often mentioned threshold of 0.5, the motive ranker did 282 true positive predictions that Stanford CoreNLP had missed, but also 1413 additional false positive errors, lowering the Precision of the CoreNLP predictions from 0.623 to 0.540.

Setting	**Precision**	**Recall**	**Fscore**	**% positive pred.**
Classification	0,1813462	0,19048546	0,1858035129	N/A
Ranker with threshold 0	0,33715987	0,33715987	0,33715987	100
Ranker with threshold 0.1	0,3820624	0,309852961	0,3421897319	81,10009281
Ranker with threshold 0.2	0,41682518	0,2355038851	0,3009645118	56,49943823
Ranker with threshold 0.3	0,44410044	0,1891944672	0,265346666	42,60172928
Ranker with threshold 0.4	0,48762023	0,1510429328	0,2306429866	30,97552635
Ranker with threshold 0.5	0,5430533	0,1004347513	0,1695180602	18,49445557
Ranker with threshold 0.6	0,5558691	0,0962337009	0,1640641341	17,31229544
Ranker with threshold 0.7	0,5646843	0,0899809577	0,1552269149	15,93473694
Ranker with threshold 0.8	0,57401526	0,0882712148	0,1530124085	15,37785159
Ranker with threshold 0.9	0,5740268	0,0878804158	0,1524253329	15,30946217

Table 4.10.: Motive prediction results - 50/50 training/test set distribution, only 2-way motive feature, no Machine Learning algorithm, direct ranking with varying thresholds

4.4.1 Conclusions

The low Precision values may suggest that this simple motive counting is not suited as a tool for coreference resolution, but the results are not as bad as they seem. With the applied maximum distance of six verbs between a pair of ports, the amount of possibilities for each coreference connection can be estimated. The minimum amount of possibilities is one, the maximum amount of connections is eighteen, as there can be six possible verbs with up to three ports each in the specified vicinity. In the given gold annotated data set, the average amount of possibilities was 7.670. This means that random guessing would result in a Precision of 0.130 whereas the motive ranker reached a Precision value of 0.337 even without using a threshold. As only one feature was used in this ranking, this was better than expected. Using more complex motive structures and utilizing their specific patterns is a very promising approach that can be combined with others systems as a completely different approach at coreference resolution.

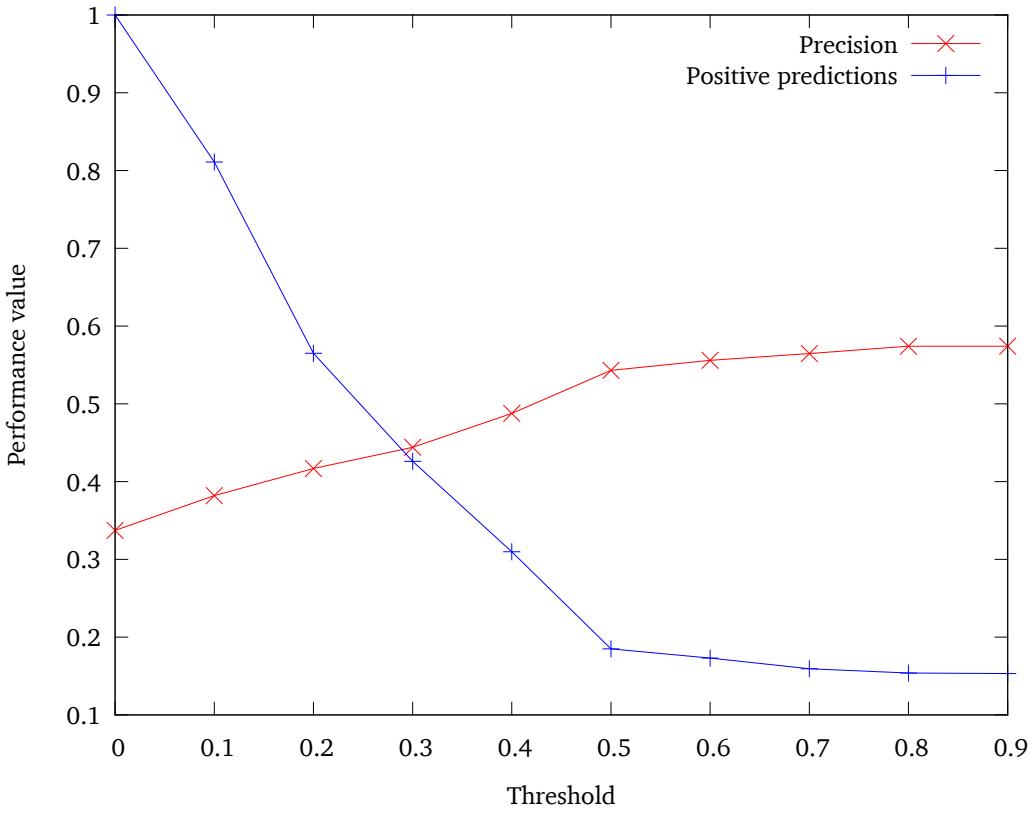

Figure 4.8.: Motive prediction results

Setting	Precision
Random baseline	0,13
Ranker without threshold	0,33715987
Ranker with 0,5 threshold	0,5430533

Table 4.11.: Motive approach versus random baseline

5 Final Conclusions

The used machine learning algorithms were able to predict coreferential connections of nominal phrases much better than the baseline of random guessing. Ranking the results and adjusting the threshold parameters allowed for more positive predictions with higher error rate, or only very few but more precise results. This system was combined with a well known existing tool for coreference resolution, and with the right threshold, the performance could be improved slightly. Although the specific performance values have to be regarded with care as the applied data model differs from the model used in common benchmarks, the positive effect of the ranker is assumed to stay true in real application environments. The resulting decision tree models showed some interesting information regarding the effects of certain lexical features, which can be used and examined in future research. The motive based approach only hinted as the possibilities that lies in utilizing reoccurring motives in natural language processing tasks.

5.1 Next Steps

Direct implementation of the Random Forest ranker in the Stanford CoreNLP system at various sieve levels should confirm the viability of this approach in a better comparable scenario. This test may be pursued in future works at the department of Computational Linguistics at Heidelberg University, which received the source code for all created applications for this purpose.

Another big field of exploration lies in the motive based approaches. Bigger motives can be utilized for more complex feature generation, or they can be used to gather additional information about similar verb and reoccurring coreference connections.

6 Acknowledgements

I want to express my sincere gratitude to my main supervisor Professor Dr. Karsten Weihe, for his time and effort in supporting my work and his constant confidence in my abilities. I also want to thank my secondary supervisor Professor Dr. Johannes Fürnkranz, for his engaging lectures that piqued my interest on the topic of machine learning, and his supportive clues to new experiments for this thesis. Special thanks go to his assistant Christian Wirth, who has been a great help in setting up the Weka experiments on the server of the Knowledge Engineering Group. Further, I want to express my appreciation to my third supervisor Professor Dr. Anette Frank and her assistant Eva Mujdricza-Maydt of the department of Computational Linguistics at Heidelberg University, for their work and support, but also their creative ideas for new approaches to the given problems. And finally, huge special thanks to my wife Michaela, who always restored my sanity and reinforced my morale.

Bibliography

[1] Bagga, Amit ; Baldwin, Breck: Entity-based cross-document coreferencing using the vector space model. In: *Proceedings of the 17th international conference on Computational linguistics-Volume 1* Association for Computational Linguistics, 1998, S. 79–85

[2] Breiman, Leo: Random forests. In: *Machine learning* 45 (2001), Nr. 1, S. 5–32

[3] Freedman, Reva ; Krieghbaum, Douglas: Relationship between Student Writing Complexity and Physics Learning in a Text-Based ITS. In: Trausan-Matu, Stefan (Hrsg.) ; Boyer, KristyElizabeth (Hrsg.) ; Crosby, Martha (Hrsg.) ; Panourgia, Kitty (Hrsg.): *Intelligent Tutoring Systems* Bd. 8474, Springer International Publishing, 2014 (Lecture Notes in Computer Science). – ISBN 978–3–319–07220–3, 656-659

[4] Lee, Heeyoung ; Peirsman, Yves ; Chang, Angel ; Chambers, Nathanael ; Surdeanu, Mihai ; Jurafsky, Dan: Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task* Association for Computational Linguistics, 2011, S. 28–34

[5] Manning, ChristopherD.: Part-of-Speech Tagging from 97Linguistics? In: Gelbukh, AlexanderF. (Hrsg.): *Computational Linguistics and Intelligent Text Processing* Bd. 6608. Springer Berlin Heidelberg, 2011, S. 171–189

[6] Mitkov, Ruslan ; Evans, Richard ; Orasan, Constantin: A new, fully automatic version of Mitkov's knowledge-poor pronoun resolution method. In: *Computational Linguistics and Intelligent Text Processing*. Springer, 2002, S. 168–186

[7] Ng, Vincent: Machine learning for coreference resolution: From local classification to global ranking. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* Association for Computational Linguistics, 2005, S. 157–164

[8] Ng, Vincent: Supervised Noun Phrase Coreference Research: The First Fifteen Years. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2010 (ACL '10), 1396–1411

[9] Quinlan, John R.: *C4. 5: programs for machine learning*. Bd. 1. Morgan kaufmann, 1993

[10] Santos, Cicero N. ; Carvalho, Davi L.: Rule and tree ensembles for unrestricted coreference resolution. In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task* Association for Computational Linguistics, 2011, S. 51–55

[11] Soon, Wee M. ; Ng, Hwee T. ; Lim, Daniel Chung Y.: A machine learning approach to coreference resolution of noun phrases. In: *Computational linguistics* 27 (2001), Nr. 4, S. 521–544

[12] Treeratpituk, Pucktada ; Giles, C L.: Disambiguating authors in academic publications using random forests. In: *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries* ACM, 2009, S. 39–48

[13] Yang, Xiaofeng ; Zhou, Guodong ; Su, Jian ; Tan, Chew L.: Coreference resolution using competition learning approach. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1* Association for Computational Linguistics, 2003, S. 176–183

List of Figures

List of Tables

A Appendix

A.1 Programm flow chart

These program steps are performed after all positive and negative examples were constructed from the input data corpus. All examples from the gold annotated documents of the Ontonotes database were transformed into one large ”ARFF” file, containing all pairings of ports that are within the same document, and within a maximum distance of six. This corresponds to the first step of conversions, as depicted as Step 1 in the program steps diagram A.1.

The ARFF format is needed for the Weka machine learning tool that was used for all experiments. Most of the attributes from the GraphML files were included in the ARFF files as String attributes.
In step 2, the full ARFF file was split into separate training and test sets. In the conducted experiments, this was an even 50 / 50 % split of the instances. It was made sure that all instances with the same source node are put in the same data subset. In the same step, the conversion to Sparse encoding was done to eliminate String features and save memory usage.

To apply the machine learning algorithms on the resulting data files, version 3.6 of the Weka toolkit from the Machine Learning Group at the University of Waikato was used. It is an open source software containing many different machine learning algorithms, filters and other helpful tools. With this toolkit two separate tree learning algorithms, the Java implementation of the C4.5 algorithm called J48 and Random Forest, were used on the data in many different experiment settings. The specific conducted experiments and their results are presented in the following chapters.

The results of this kind of Weka experiment were two files. The first file contained the model that was constructed during the learning process, and will be used to predict new instances of data. In the case of the applied tree algorithms, the model file contained one or more decision trees. These files are serialized in a Weka specified format, but they can be analysed or applied to new data by using the Weka toolkit. The analysis and the associated results can be found in chapter 3.5.

The second file that emerged from a Weka experiment was an output file that included the predictions of the provided test data. These predictions were created by applying the model from the learning process on the test set. For every instance in the test set, the output file contains one line of text that consists of a consecutive number of the instance, its real class attribute, the predicted class attribute, and the calculated probability that the instance belongs to the predicted class. Various programs were created to analyse these output prediction files.

A.2 GraphML example file

This GraphML example file shows the general format of the used input data. The top part contains definitions of all attributes, followed by all verb nodes with attached port expressions. After every node has been listed, the coreference edges connect two ports each by using their unique identifier.
This example shows one node with two ports, and two edges.

```
1 <?xml version="1.0" encoding="UTF-8"?>

3 <!-- keys for graph -->
  <key id="docId" for="graph" attr.name="docId" attr.type="string"/>
5 <key id="text" for="graph" attr.name="text" attr.type="string"/>
```

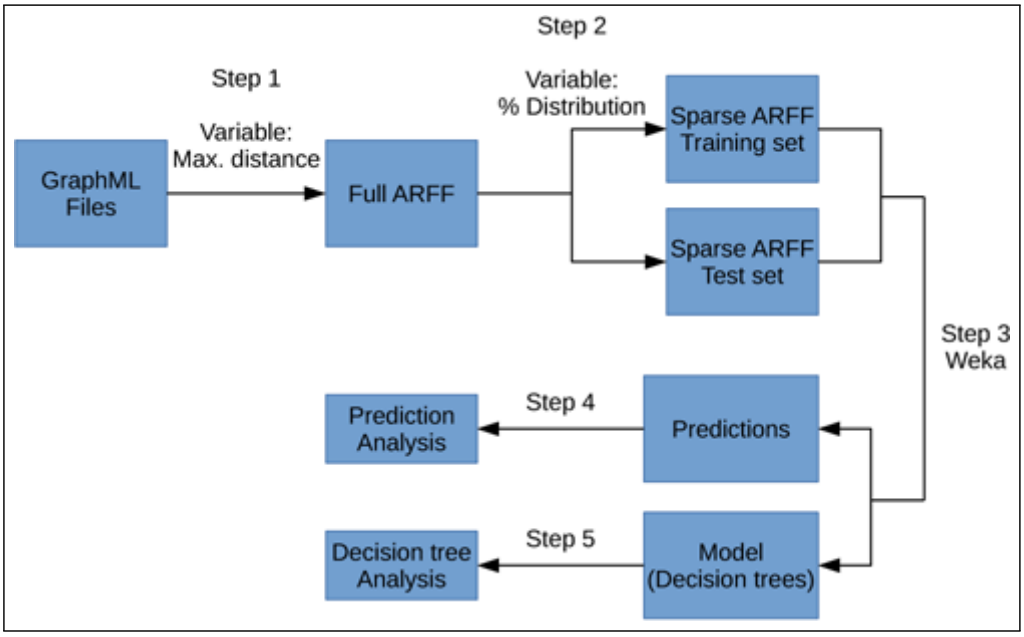

Figure A.1.: Program flow chart

```
<!-- keys for node -->
<key id="sentIdx" for="node" attr.name="sentIdx" attr.type="int"/>
<key id="tokIdx" for="node" attr.name="tokIdx" attr.type="int"/>
<key id="tok" for="node" attr.name="tok" attr.type="string"/>
<key id="lemma" for="node" attr.name="lemma" attr.type="string"/>
<key id="pos" for="node" attr.name="pos" attr.type="string"/>
<key id="verbRels" for="node" attr.name="verbRels" attr.type="string"/>
<key id="stanfLabel" for="node" attr.name="stanfLabel" attr.type="string"/>
<key id="voice" for="node" attr.name="voice" attr.type="string"/>

<!-- keys for port -->
<key id="tokIdx" for="port" attr.name="tokIdx" attr.type="int"/>
<key id="tok" for="port" attr.name="tok" attr.type="string"/>
<key id="lemma" for="port" attr.name="lemma" attr.type="string"/>
<key id="pos" for="port" attr.name="pos" attr.type="string"/>
<key id="ner" for="port" attr.name="ner" attr.type="string"/>
<key id="gramFct" for="port" attr.name="gramFct" attr.type="string"/>
<key id="stanfLabel" for="port" attr.name="stanfLabel" attr.type="string"/>

<!-- keys for edge -->
<key id="toGramFct" for="edge" attr.name="toGramFct" attr.type="string"/>
<key id="toNodeId" for="edge" attr.name="toNodeId" attr.type="int"/>
<key id="sentIdx" for="edge" attr.name="sentIdx" attr.type="int"/>
<key id="tokIdx" for="edge" attr.name="tokIdx" attr.type="int"/>
<key id="isSameArg" for="edge" attr.name="isSameArg" attr.type="boolean">
  <default>false</default>
</key>
<key id="verbRel" for="edge" attr.name="verbRel" attr.type="string"/>
<key id="repres" for="edge" attr.name="repres" attr.type="string"/>
<key id="corefChain" for="edge" attr.name="corefChain" attr.type="int"/>

<graph id="bc/cctv/00/cctv_0000.000" edgedefault="directed">
  <data key="docId">bc/cctv/00/cctv_0000.000</data>
  <node id="n0">
    <data key="sentIdx">0</data>
```

```
    <data key="tokIdx">13</data>
    <data key="tok">looking</data>
    <data key="lemma">look</data>
    <data key="pos">VBG</data>
    <data key="verbRels">[10chain222, 10chain350, 12chain183, 12chain188, 12chain278, 6
    chain225, 6chain306, 6chain566, 8chain246, 8chain264, 8chain460, ent, tmp]</data>
    <data key="stanfLabel">rcmod</data>
    <data key="voice">active</data>
    <port name="pD0">
      <data key="gramFct">D</data>
      <data key="tokIdx">8</data>
      <data key="tok">that</data>
      <data key="lemma">that</data>
      <data key="pos">WDT</data>
      <data key="ner">O</data>
      <data key="stanfLabel">dobj</data>
    </port>
    <port name="pS1">
      <data key="gramFct">S</data>
      <data key="tokIdx">9</data>
      <data key="tok">people</data>
      <data key="lemma">people</data>
      <data key="pos">NNS</data>
      <data key="ner">O</data>
      <data key="stanfLabel">nsubj</data>
    </port>
  </node>

<!-- more nodes -->

  <edge id="e0" source="n14" target="n3" sourceport="pS18" targetport="pS4">
    <data key="verbRel">[noRel]</data>
    <data key="repres">park</data>
    <data key="corefChain">2</data>
  </edge>
  <edge id="e1" source="n19" target="n14" sourceport="pD24" targetport="pS18">
    <data key="verbRel">[noRel]</data>
    <data key="repres">park</data>
    <data key="corefChain">2</data>
  </edge>

<!-- more edges -->

</graph>
</graphml>
```

A.3 ARFF example file

This example file shows the normal ARFF format with the full feature set and a few example instances.

```
@RELATION Koreferenz
@ATTRIBUTE Source string
@ATTRIBUTE SrcLemma string
@ATTRIBUTE SrcPos string
@ATTRIBUTE SrcStanfLabel string
@ATTRIBUTE SrcVoice string
@ATTRIBUTE SrcPortGramFct string
@ATTRIBUTE SrcBindRahmen string
@ATTRIBUTE SrcPortPos string
@ATTRIBUTE SrcPortNer string
@ATTRIBUTE TarLemma string
@ATTRIBUTE TarPos string
@ATTRIBUTE TarStanfLabel string
@ATTRIBUTE TarVoice string
@ATTRIBUTE TarPortGramFct string
@ATTRIBUTE TarBindRahmen string
@ATTRIBUTE TarPortPos string
@ATTRIBUTE TarPortNer string
@ATTRIBUTE Distance numeric
@ATTRIBUTE LemmaGleich {0,1}
@ATTRIBUTE TokenGleich {0,1}
@ATTRIBUTE PortLemmaGleich {0,1}
@ATTRIBUTE PortTokenGleich {0,1}
@ATTRIBUTE BindRahmenGleich {0,1}
@ATTRIBUTE StanfLblGleich {0,1}
@ATTRIBUTE PortGrmFctGleich {0,1}
@ATTRIBUTE AnzahlSrcVerbRels numeric
@ATTRIBUTE AnzahlTarVerbRels numeric
@ATTRIBUTE class {0,1}
@DATA
{"1_0_S","start","VBD","root","active","S","S","NN","O","look","VBG","rcmod","active","D","SD","WDT","O",1,0,0,0,0,0,0,0,9,13,0}
{"1_0_S","start","VBD","root","active","S","S","NN","O","look","VBG","rcmod","active","S","SD","NNS","O",1,0,0,0,0,0,0,1,9,13,0}
{"2_0_S","cause","VBD","root","active","S","S","NNS","O","look","VBG","rcmod","active","D","SD","WDT","O",2,0,0,0,0,0,0,0,55,13,0}
{"2_0_S","cause","VBD","root","active","S","S","NNS","O","look","VBG","rcmod","active","S","SD","NNS","O",2,0,0,0,0,0,0,1,55,13,0}
{"2_0_S","cause","VBD","root","active","S","S","NNS","O","start","VBD","root","active","S","S","NN","O",1,0,0,0,0,1,1,1,55,9,0}
{"3_0_S","open","VB","root","active","S","S","NN","O","look","VBG","rcmod","active","D","SD","WDT","O",3,0,0,0,0,0,0,0,0,22,13,0}
{"3_0_S","open","VB","root","active","S","S","NN","O","look","VBG","rcmod","active","S","SD","NNS","O",3,0,0,0,0,0,0,0,1,22,13,0}
{"3_0_S","open","VB","root","active","S","S","NN","O","start","VBD","root","active","S","S","NN","O",2,0,0,0,0,1,1,1,22,9,0}
{"3_0_S","open","VB","root","active","S","S","NN","O","cause","VBD","root","active","S","S","NNS","O",1,0,0,0,0,1,1,1,22,55,0}
```

A.4 Sparse ARFF example file

This example file shows the Sparse ARFF format with a heavily abbreviated feature set and a few example instances.

```
@RELATION Koreferenz_Sparse
@ATTRIBUTE Source string
@ATTRIBUTE SrcLemma_000 {0,1}
@ATTRIBUTE SrcLemma_001 {0,1}
@ATTRIBUTE SrcLemma_002 {0,1}
@ATTRIBUTE SrcLemma_003 {0,1}
@ATTRIBUTE SrcLemma_004 {0,1}
@ATTRIBUTE SrcLemma_005 {0,1}
@ATTRIBUTE SrcLemma_006 {0,1}
@ATTRIBUTE SrcLemma_007 {0,1}
...
@ATTRIBUTE TarPortNer_017 {0,1}
@ATTRIBUTE TarPortNer_018 {0,1}
@ATTRIBUTE Distance numeric
@ATTRIBUTE LemmaGleich {0,1}
@ATTRIBUTE TokenGleich {0,1}
@ATTRIBUTE PortLemmaGleich {0,1}
@ATTRIBUTE PortTokenGleich {0,1}
@ATTRIBUTE BindRahmenGleich {0,1}
@ATTRIBUTE StanfLblGleich {0,1}
@ATTRIBUTE PortGrmFctGleich {0,1}
@ATTRIBUTE AnzahlSrcVerbRels numeric
@ATTRIBUTE AnzahlTarVerbRels numeric
@ATTRIBUTE class {0,1}
@DATA
{0 "1_0_S",320 1,3311 1,3411 1,3436 1,3438 1,3442 1,3448 1,3496 1,4225 1,6815 1,6850
1,6939 1,6942 1,6944 1,6976 1,6998 1,7005 1,7006 0,7007 0,7008 0,7009 0,7010 0,7011 0,7012
 0,7013 9,7014 13,7015 0}
{0 "1_0_S",320 1,3311 1,3411 1,3436 1,3438 1,3442 1,3448 1,3496 1,4225 1,6815 1,6850
1,6939 1,6941 1,6944 1,6959 1,6998 1,7005 1,7006 0,7007 0,7008 0,7009 0,7010 0,7011 0,7012
 1,7013 9,7014 13,7015 0}
{0 "2_0_S",1836 1,3311 1,3411 1,3436 1,3438 1,3442 1,3457 1,3496 1,4225 1,6815 1,6850
1,6939 1,6942 1,6944 1,6976 1,6998 1,7005 2,7006 0,7007 0,7008 0,7009 0,7010 0,7011 0,7012
 0,7013 55,7014 13,7015 0}
{0 "2_0_S",1836 1,3311 1,3411 1,3436 1,3438 1,3442 1,3457 1,3496 1,4225 1,6815 1,6850
1,6939 1,6941 1,6944 1,6959 1,6998 1,7005 2,7006 0,7007 0,7008 0,7009 0,7010 0,7011 0,7012
 1,7013 55,7014 13,7015 0}
{0 "2_0_S",1836 1,3311 1,3411 1,3436 1,3438 1,3442 1,3457 1,3496 1,3820 1,6814 1,6914
1,6939 1,6941 1,6945 1,6951 1,6998 1,7005 1,7006 0,7007 0,7008 0,7009 0,7010 1,7011 1,7012
 1,7013 55,7014 9,7015 0}
```