Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master Thesis

Comparative Argument Mining

Mirco Franzek
Matrikelnummer: 6781911
MIN-Fakultät
Fachbereich Informatik
Studiengang: Informatik

Erstgutachter: Prof. Dr. Chris Biemann
Zweitgutachter: Dr. Alexander Panchenko

Contents

1. Introduction: An Open-Domain Comparative Argumentative Machine

This thesis discusses the topic of *comparative argument mining*. Comparative argument mining is a subfield of *argument mining*, a recent research topic in natural language processing.

The goal is to develop a system which is able to decide if a given sentence compares two known objects and, if it does, which object wins the comparison. For instance, given the sentence *"In my mind, Python is better than Java!"*, the system should answer that the sentence is comparative and that Python won the comparison.

Such a system can be useful in several ways. First, it enables machines to understand statements of such sentences. Secondly, this knowledge can be used in (commercial) applications, like opinion mining or online comparison portals. As presented in Section 2.2, these portals usually rely on information from databases. The system presented in this thesis could be used to generate new knowledge automatically from forum posts, comments, tweets and the like. Furthermore, this new knowledge would include the thoughts and opinions of their authors. This stands in contrast to the pure factual information from databases. For instance, the system could find out that many people complain about the telephone support of insurance company X, while they praise the support of company Y - an information rarely found on comparison portals.

The thesis is structured into three main parts. The first part discusses recent publications in argument mining and explains the needed concepts from natural language processing and machine learning.

Because comparative argument mining is a novel field, no suitable data set for this task currently exists. The second part describes the creation of a data set with data crawled from the web and crowdsourcing methods to label the data. The result is a data set with 7199 sentences, containing comparisons of 271 distinct object pairs. Each sentence is annotated with one of three classes (`BETTER`, `WORSE`, `NONE`) which reflects whether the sentence is comparative and whether the first mentioned object in the sentence won the comparison.

The third part uses this data set to train machine learning models on several feature representations in order to predict the correct class. As the final evaluation, all features are tested on held-out data.

2. Background

2.1. Related Work

In the following, publications on argument mining are presented. If appropriate, the f1 scores achieved in these publications are presented as well. It must be noted that the f1 scores are not comparable with each other.

A general introduction on the research topic argument mining was given in [Lippi and Torroni, 2016]. The authors introduced five dimensions to describe argument mining problems: granularity of input, the genre of input, argument model, the granularity of target and goal of analysis. Furthermore, the typical steps of argument mining systems were described. First, the input is divided into argumentative (e.g. claim and premise) and non-argumentative parts. This step was described as a classification problem. Secondly, the boundaries of the argumentative units must be identified; this was understood as a segmentation problem. Thirdly, the relations between argumentative units must be identified. For instance, claims and premises might be connected with a *support* or a *attack* relation.

A system, which is capable of recognising comparative sentences and their components such as the compared entities, the property which is used to compare the entities and the direction of comparison, was presented in [Fiszman et al., 2007]. The evaluation showed that the outcome has a high quality (f1 score of 0.81). However, the presented system was specific to the domain of studies on drug therapy. The system used patterns generated from hand-selected sentences, as well as domain knowledge. Therefore, the methods cannot be transferred easily to the problem of this thesis.

In [Park and Blake, 2012], the authors presented another domain-specific approach on argumentative sentence detection. The problem was formulated as a binary classification task. As in [Fiszman et al., 2007], the features were tailored for medical publications. The authors conducted a pilot study with 274 comparison sentences extracted from abstracts and 164 comparison sentences from full text articles. The sentences were analysed in order to extract 35 features (six lexical features, 27 syntactic features and two miscellaneous features). For instance, a lexical feature capturing the appearance of *"versus"* or *"vs."* was developed, while a miscellaneous feature checked if the subject of the comparison is in plural.

A recent publication on comparative argument mining is [Gupta et al., 2017], in which a set of rules for the identification of comparative sentences (and the compared entities) was derived from syntactic parse trees. With this set of rules, the authors achieved a

f1 score of 0.87 for the identification of comparative sentences. The rules were obtained from 50 abstracts of biomedical papers. Such being the case, they are domain dependent.

The challenges occurring while processing texts from social media were described in [Šnajder, 2017]. In addition to the noisiness of text, missing argument structures and poorly formulated claims were mentioned. It is expected that the text used in this thesis will have the same shortcomings. Additionally, [Šnajder, 2017] emphasized that analyzing social media texts can provide reasons behind opinions.

In addition to the challenges mentioned above, [Dusmanu et al., 2017] also pointed to the specialized jargon in user-generated content like hashtags and emoticons. With this in mind, [Dusmanu et al., 2017] classified tweets about the "*Brexit*" and "*Grexit*" either as argumentative or as non-argumentative. In addition to the features that were used in publications mentioned in this section, new features covering hashtags and sentiment were added. They achieved a f1 score of 0.78 (using logistic regression) for the classification. It must to be mentioned that the data set is small (1887 tweets) and the domain is rather specific.

Publications dealing with the identification of argument structures are of relevance for this thesis, as they provide valuable insights on the suitability of features and algorithms.

The authors of [Aker et al., 2017] summarised and compared features used in other publications for identification of argumentative sentences. In addition to the algorithms used in the publications, a convolutional neural network (as described in [Kim, 2014]) was tested. Two existing corpora and six different classification algorithms were used. The comparison resulted in the insight that structural features and random forests worked the best.

A two-step procedure to identify components of arguments (such as claim and premise) and their relationships (like "premise A supports claim B") is presented in [Stab and Gurevych, 2014]. The identification step is formulated as a multi-class classification. For the identification of argumentative components, a f1 score of 0.72 was reported.

How different datasets represent the argumentative unit of a claim was analysed in [Daxenberger et al., 2017]. After an analysis of the data sets and their annotation scheme, the authors conducted two experiments. In the first one, each learner was trained and evaluated (10-fold cross-validation) on each dataset one after another. On average, the macro f1 score for the identification of claims was 0.67 (all results ranging from 0.60 to 0.80). No significant difference between the results of logistic regression and the neural networks was found. In isolation, lexical features, syntactical features and word embeddings were most helpful. Structural features turned out to be the weakest. The second experiment was conducted in a cross-domain fashion. For each pair of data sets, one was used as the training set and the other one as the test set. The average macro f1 score was 0.54. In this scenario, the best feature combination outperformed all neural models. The

authors assumed that there might not be enough training data for the neural models. As a last point, the authors noted that all claims share at least some lexical clues.

The role of discourse markers in the identification of claims and premises were discussed in [Eckle-Kohler et al., 2015]. A discourse marker is a word or a phrase which connects discourse units. For instance, the word "as" can show a relation between claim and premise: "As the students get frustrated, their performance generally does not improve". A similar function is expected for words like better, worse or because in this thesis. The authors showed that discourse markers are good at discriminating claim and premises. If claim and premise are merged into one class "argumentative", this can be used to identify argumentative sentences. The f1 score is not presented, but the accuracy is between 64.53 and 72.79 percent.

2.2. Domain-Specific Comparative Systems

Comparison portals are a possible application for comparative argument mining. Many comparison portals can be found on the internet. It is not unusual to see a television commercial of these comparison portals, which suggests that they are used frequently every day.

Most portals are specific to a few domains and a subset of properties, for example, car insurances and their price. Comparisons are only possible between objects of the domains and predefined properties. Source of the data is usually databases. Humans are involved in gathering, entering and processing the data.

Comparison portals only compare and deliver facts. Because of that, they can only hint to choose *X* over *Y* based on the facts collected. However, an insurance company *X* might be the best in the comparison (for instance, best price), while the internet is full of complaints about its poor service.

Examples of comparative portals are *Check24.de*[1], *Verivox.de*[2], *Idealo.de*[3], *GoCompare.com*[4], and *Compare.com*[5] just to name a few.

As an example, Check24.de is able to compare a wide variety of different objects like several insurance companies, credit cards, energy providers, internet providers, flights, hotels and car tires. After the user entered some details (based on the object type), Check24.de shows a ranking of different providers. The user can choose different properties to re-rank the list. For instance, to compare different DSL providers, the user has to enter the address , how fast the internet should be and if telephone and television are wanted as well (see Figure 2.2.1). The user can then sort the results by price, speed, and grade (Figure 2.2.2).

[1] `https://check24.de` (checked: 13.04.2018)
[2] `https://verivox.de` (checked: 13.04.2018)
[3] `https://idealo.de` (checked: 13.04.2018)
[4] `https://gocompare.com` (checked: 13.04.2018)
[5] `https://compare.com` (checked: 13.04.2018)

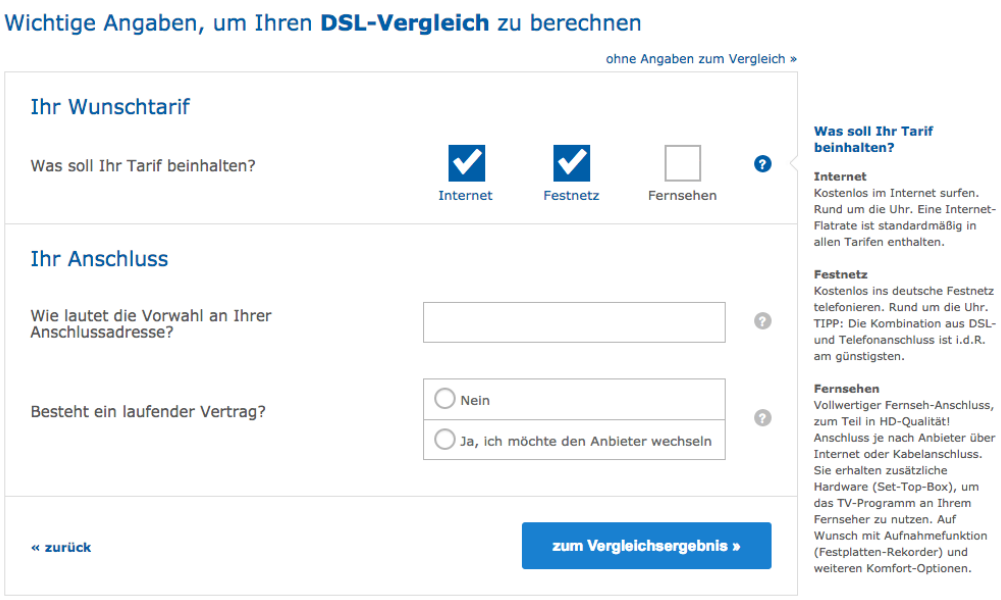


Figure 2.2.1.: Check24.de asks for some data before comparing different DSL contracts

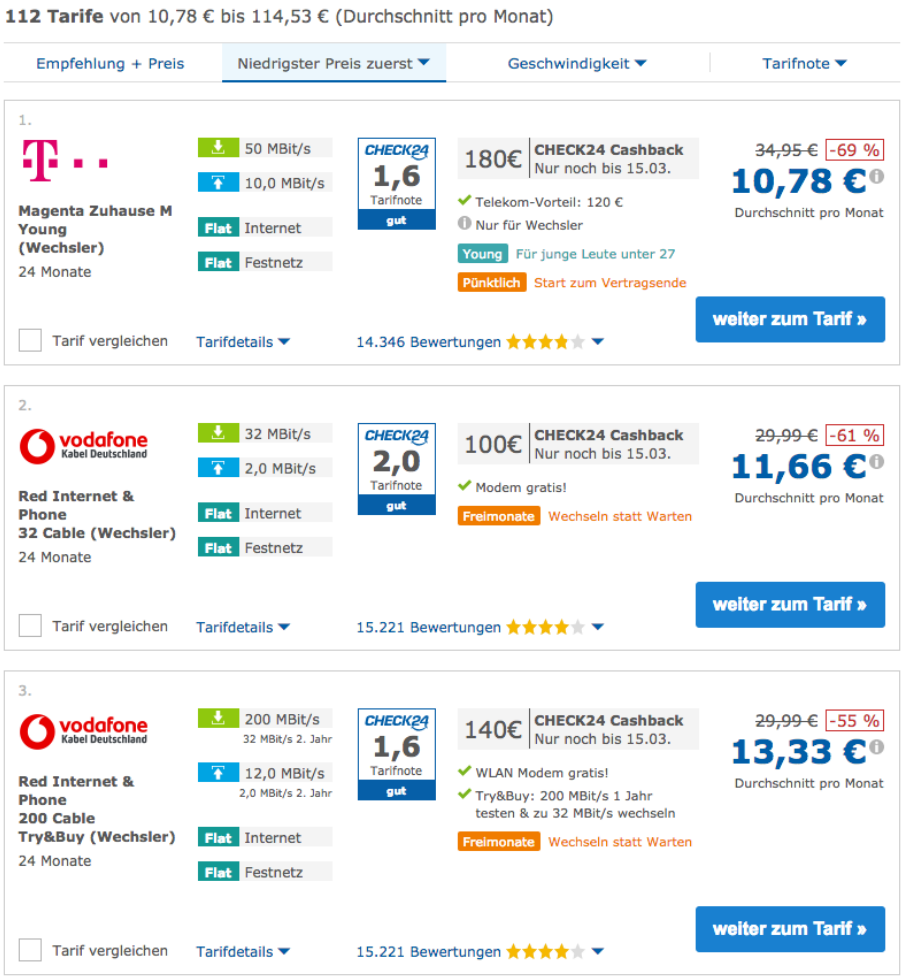


Figure 2.2.2.: Check24.de DSL contract comparison result. The contracts can be sorted by domain-specific criteria.

The other sites work similarly. All in all, they provide more of a ranking than a comparison.

Another interesting type of website are question answering portals like *Quora.com*[6] or *GuteFrage.net*[7]. Although comparisons are not their primary goal, a lot of comparative questions are present on those sites. On Quora.com, more than 2.380.000 questions have the phrase "*better than*" in their title. If "*Ruby*" and "*Python*" are added, 10.100 questions remain.[8] Same is true for the German site GuteFrage.net, though, the numbers are smaller than on Quora.com.[9]

More interestingly are systems which can compare any objects on arbitrary properties, like *Diffen.com*[10] and *Versus.com*[11].

Versus.com aggregates freely available data sources like Wikipedia or official statistic reports. For example, the comparison of *"Hamburg vs. Berlin"* uses Wikipedia for the number of universities, `worldstadiums.com` for the availability of sport facilities and the Economist for the Big Mac Index. Presumably, human processing is involved as the possible comparisons are limited. For instance, a comparison of Hamburg and Darmstadt is not possible as Darmstadt is not available on Versus.com[12]. Likewise, *"Ruby vs. Python"* is not possible, Versus.com suggests to compare *"Rome vs. Pyongyang"* instead. Although Versus.com shows how many users liked the objects, it does not give a clear statement which one is better. For instance, it is not possible to check automatically whether Hamburg or Berlin is better for a short city trip. The user must manually search all valid properties like the number of museums, theaters, the price of public transport tickets and so on.

Similar to Versus.com, Diffen.com aggregates different data sources (see Figure 2.2.3 and Figure 2.2.4). All in all, the aggregated information is similar to Versus.com. The comparison is also tabular. Besides the automatically aggregated data, users can add information on their own. Diffen.com does not enforce any restrictions on the objects of comparison, but it faces the same problem as Versus.com as objects are missing. A comparison between Darmstadt and Hamburg is likewise not possible: all cells for Darmstadt in the table are empty.

Neither Versus.com nor Diffen.com provides a comprehensible reason why an object is better than another one. They merely aggregate facts and bring them face to face. Despite the aggregation approach of both systems, many meaningful comparisons are not possible or not helpful (like *"Hamburg vs. Darmstadt"*, *"Java vs. C#"*, *"Dr Pepper vs. Orange Juice"*). Also, the user can not define the properties for the comparison. The sites provide

[6] `https://quora.com` (checked: 13.04.2018)
[7] `https://gutefrage.net` (checked: 13.04.2018)
[8] Checked via Google on 11.12.2017. Search phrase: `"better than" site:quora.com` and `ruby python "better than" site:quora.com`
[9] 334.000 for `"besser als" site:gutefrage.net` and 78 for `ruby python "Besser als" site:gutefrage.net`
[10] `https://diffen.com` (checked: 13.04.2018)
[11] `https://versus.com` (checked: 13.04.2018)
[12] Checked on 14.05.2018

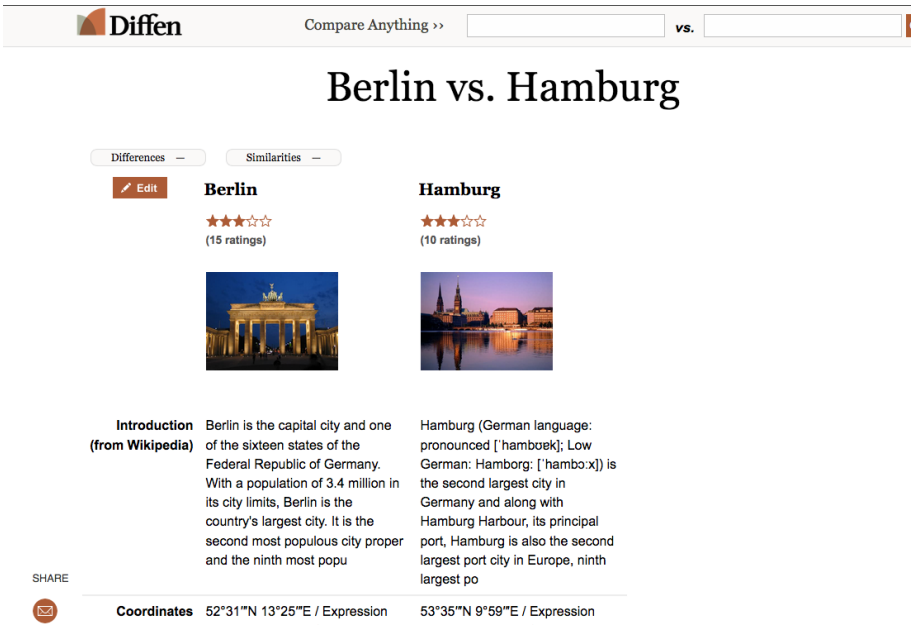


Figure 2.2.3.: The comparison of *"Hamburg vs. Berlin" on Diffen.com*

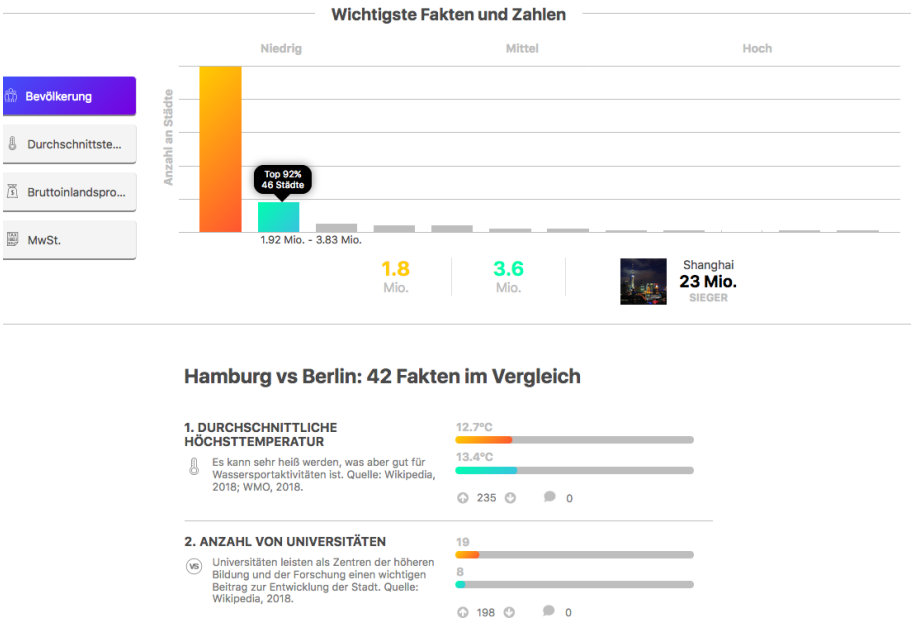


Figure 2.2.4.: The comparison of *"Hamburg vs. Berlin" on Versus.com*

every information available for the objects. For instance, Versus.com shows 42 properties for "Hamburg vs. Berlin" but only 35 for "Hamburg vs. Munich".

To summarise, a lot of different comparison portals exist and are widely used. Especially the domain-specific portals do a good job, but inflexibility dearly buys the performance. First, the portals can only compare objects on predefined properties. Second, the data acquisition is not fully automatic. Domain-unspecific systems are good at aggregating information but do not provide a reasonable explanation to prefer X over Y.

Adding information like comments and product reviews can enrich the comparison with reasons and opinions, such as *"Ruby is easier to learn than C"* or *"Python is more suitable for scientific applications than Erlang as many libraries exist"*.

2.3. Machine Learning Methods

The goal of Chapter 4 was to find the most appropriate category for each sentence of the data set created in Chapter 3. This was understood as a classification problem and solved using several machine learning methods. The following sections describe these methods. The descriptions are based on [Mitchell, 1997], [Friedman et al., 2009] and [Goodfellow et al., 2016].

2.3.1. Performance Measures

Several ways exist to evaluate classification models. A simple measure is *accuracy*, which is defined as the fraction of correct predictions:

$$Acc = \frac{\text{\# correct predictions}}{\text{total \# of predictions}}$$

Accuracy is only suitable if all classes have the same size. For instance, a data set with 950 positive examples and 50 negative examples will get an accuracy of 95% if it always predicts the positive class.

Precision, *recall* and *f1 score* are measures to check the classification performance on data sets with imbalanced classes. The measures are calculated per class.

The precision of a classifier with regards to class *c* is defined as:

$$\text{P(c)} = \frac{\text{\# of true positives}}{\text{\# true positives + \# false positives}}$$

Precision is the ratio between correctly predicted examples for class *c* and all predicted examples of class *c*.

The recall of a classifier with regards to class *c* is defined as:

$$\text{R(c)} = \frac{\text{\# of true positives}}{\text{\# true positives + \# false negatives}}$$

If only one out of 1000 examples was classified as c, and the classification was correct, the precision for c is one, while the recall is near zero. Likewise, if all examples are classified as c, the recall is one, while the precision is zero.

The f1 score balances precision and recall. It is defined as the harmonic mean:

$$f_1(c) = \frac{2\mathrm{PR}}{\mathrm{P+R}}$$

2.3.2. Neural Networks

Neural networks are a powerful, widely used machine learning method for classification and regression. The basic building block of neural networks is the *neuron* (also called *cell*). A neuron takes m input values and produces n output values:

$$\vec{y} = \phi\left(\sum_{i=0}^{m} w_i x_i\right)$$

where w_i is a weight, ϕ is the activation function and $\vec{y}$ is the a vector of size n. The weights are the trainable parameters of a neuron. The *perceptron*, presented in [Rosenblatt, 1958], is the simplest form of a neuron. The perceptron ϕ is defined the *heaviside step function*, which returns one if the sum is greater zero and zero otherwise. Hence, a single perceptron is a linear, binary classifier.

The *multilayer perceptron* is a neural network build with multiple layers of neurons inspired by the perceptron. In contrast to the perceptron, a differentiable function is used as the activation function ϕ (for example *sigmoid* or *rectied linear activation function*[13]). This is required because the backpropagation algorithm used for updating the weights makes use of derivatives. A detailed description on backpropagation is given in Section 4.5.2 of [Mitchell, 1997].

Figure 2.3.1 shows an example with one hidden layer of size three. The neurons of the first layer (input layer) do not calculate anything, each neuron outputs the associated input value. These values are fed into each neuron of the second (hidden) layer, which produces an output as described above. The output is then fed into the neuron in the last layer (the output layer). The interpretation of the output depends on the used activation function. For binary classification, the network in Figure 2.3.1 would use the *sigmoid* function. The output value is then interpreted as the probability of the input $\vec{x}$ to belong to the positive class. For multi-class classification, the output layer would have as many neurons as classes are present, and use the *softmax* function as the activation function. Neural networks as in Figure 2.3.1 are called *feed-forward (artificial) neural networks (ANN)*. The data flows sequentially from the input layer through the hidden layers until they reach the output layer.

In contrast to this, *recurrent neural networks (RNN)* may contain loops. This means that the output of layer h at time t is part of the input of layer h at time $t+1$. Figure 2.3.2 shows

[13] see page 170 of [Goodfellow et al., 2016]

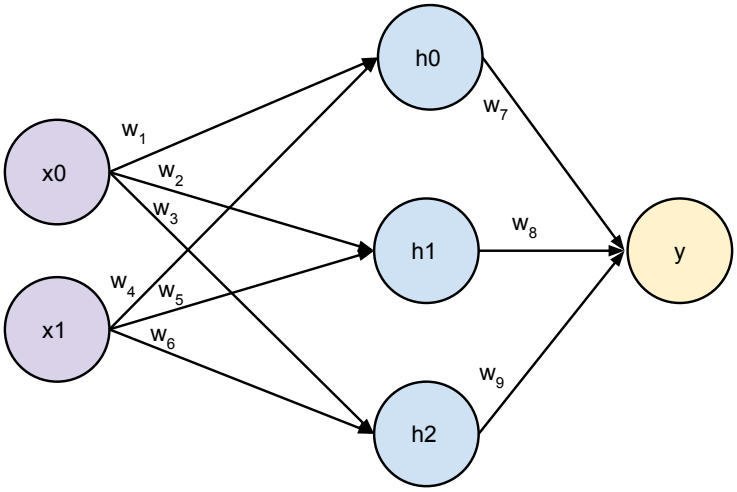


Figure 2.3.1.: Multilayer perceptron with one hidden layer. The network takes two inputs and produces one output. Larger input and output layers as well as more and large hidden layers are possible as well.

this for three time steps.

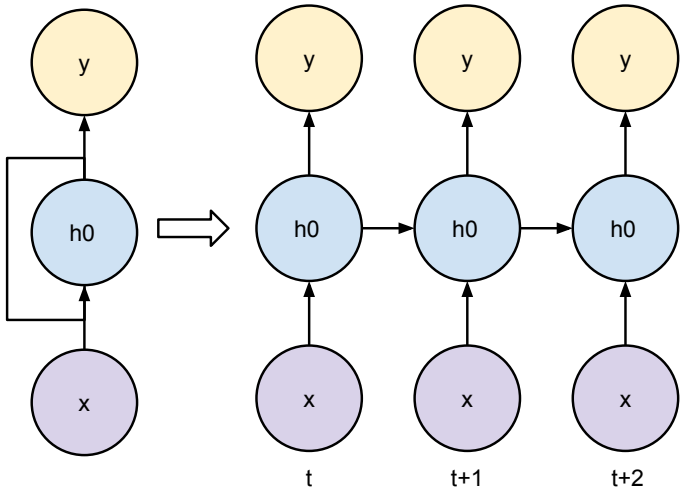


Figure 2.3.2.: Schematic view of an RNN. The right side shows the RNN unfolded for two time steps.

In this way, the network can take information from the past into account. This is useful if the inputs to the network are sentences. Each word in a sentence can be seen as a time step. Information about preceding words is often important to understand a sentence correctly. However, the simple RNN is not good at learning dependencies between words which are far away from each other in the sentence.

A frequently used type of recurrent neurons are *long short-term memory (LSTM)* blocks as presented in [Hochreiter and Schmidhuber, 1997]. LSTMs have a state, which enables them to learn long distance dependencies (up to 1000 time steps). On each time step, gates decided based on the current input (time *t*) and the previous input (time *t-1*) if data should be read, written or deleted from the state. The gates work similar to neurons as

each gate has a own set of weights and an activation function.

2.3.3. Decision Trees and Gradient Boosting

Decision trees are machine learning methods that can be used to classify[14] a data set by the repeated application of simple rules. Each rule splits the data into a subset, on which further rules are applied until no more rules are available. This is the same as traversing a tree until a leaf node is reached. An example tree for the binary classification of a sentence (comparative or not) is shown in Figure 2.3.3. The rules and their order are learned from the data set. The first rule (the root of the tree) should split on the attribute which is most useful for classification, while the second level of the tree should use the second best attributes and so on.

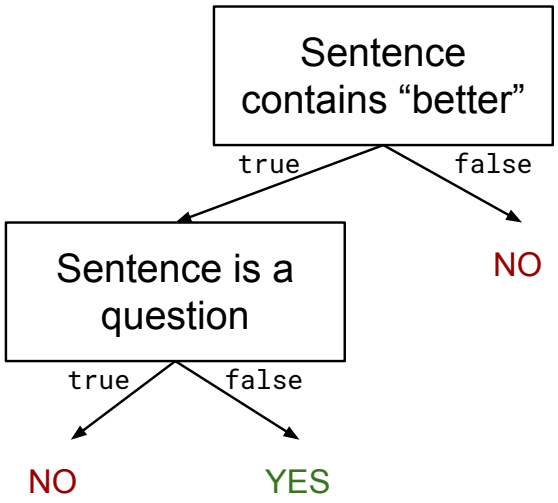


Figure 2.3.3.: Simple decision tree to check if a sentence is comparative or not.

Several algorithms like *ID3* (presentend in [Quinlan, 1986]) or *CART* (presented in [Breiman et al., 1984]) can be used to create a decision tree. A general introduction to decision tree methods is given in Chapter 3 of [Mitchell, 1997].

Boosting is a technique to combine a set of weak learners into one good learner. The predictions of a weak learner are only slightly better than random guessing. For instance, a weak classifier should get an accuracy slightly over 50% for a data set with 100 positive and 100 negative examples.

The main idea behind *gradient boosting* is to fit the weak learner sequentially on modified versions of the data. In the end, the predictions of all weak learners G_m are combined to produce the final prediction:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

[14] Decision trees can also be used for regression

The values for α_m are computed by the algorithm and determine how much the weak learner G_m contributes to the prediction.

Boosting can be used with various machine learning algorithms and is suitable for regression as well as classification tasks. The boosting method used in this thesis is *gradient boosting*[15] with decision trees as learners. In gradient boosting, G_{m+1} is fitted on the residuals of G_m. Thus, each tree tries to improve on the training examples on which the previous learner was weak on.

A frequently used implementation of gradient boosted decision trees is *XGBoost*[16], as presented in [Chen and Guestrin, 2016].

2.4. Vector Representations for Documents

Because many machine learning algorithms, especially neural ones, work with numeric values as input, text must be transformed into a numerical representation. Several known methods are described in the following sections.

2.4.1. Bag-of-words and Bag-of-ngrams

The *bag-of-words* model is a simple vector representation for documents. All words in the corpus are collected into a vocabulary $\mathcal{V}$. A document j is represented by a vector $\vec{d_j}$ of size $|\mathcal{V}|$, where $\vec{d}_{j,i}$ is the frequency of word $\mathcal{V}_i$ in the document j (the *term frequency* $\text{tf}(i, j)$).

The model is fast to calculate but does not take any sequence, word frequency or grammar information into account. For instance, "*the*" appears in almost every English text, while *"psychology"* is seldom. However, seldom words are more important for the meaning of a sentence as frequent ones. This is not reflected in $\vec{d_j}$, as "*the*" is likely to get a higher value than *"psychology"*. Another problem is that the vectors are as long as the vocabulary (typically thousands of words) and sparse, which adds more parameters to learn.

The first problem can be reduced by using n-grams (hence, *bag-of-ngrams*) instead of words. The vocabulary $\mathcal{V}$ will contain all sequences of n consecutive words appearing in the corpus instead of single words. In this way, some sequence information is kept. The second problem can be solved by removing very frequent words like *"the"* or *"can"* (often called stop words) and by using a weighting function for the remaining words. A commonly used weighting function is *term frequency, inverse document-frequency (tf-idf)*. With tf-idf, $\vec{d}_{j,i}$ is calculated as:

$$\vec{d}_{j,i} = \text{tf}_{i,j} \times \log \frac{N}{n_i}$$

[15] A detailed description is given in chapter ten of [Friedman et al., 2009].
[16] `https://github.com/dmlc/xgboost` (checked 14.05.2018)

where N is the total number of documents and n_i is the number of documents in which the word (or n-gram) i appears. Tf-idf takes the frequency of words into account. Frequent words which appear in many documents will get a low weight, while words which appear seldomly get a high weight.

The other problems, sparsity and length, remain. They are solved in other vector representations.

2.4.2. Mean Word Embeddings

Word embeddings are dense, low-dimension vector representations of words. They are learned by neural networks. The basic idea is to train an neural network on a suitable task. The weight matrix (*embedding matrix*) of a defined hidden layer contains the word embeddings after the training.

The matrix is initialized with random values. After the network was trained, each column represents one word in the vocabulary $\mathcal{V}$ and each embedding has as many components as the hidden layer has neurons. This leads to word vectors which are much smaller than $|\mathcal{V}|$.

A method to learn word embeddings with feed-forward neural networks was presented in [Bengio et al., 2003]. The network was trained to predict the most likely next word given a number of preceding context words. This exploits the *distributational hypothesis* ([Harris, 1954]) which states that words which appear in similar contexts have a similar meaning. For instance, the network will see that the context "*feed the*" is often followed by words like "*cat*" or "*dog*", but never by "*chair*". In the end, the similarity between the learned vectors of related words (like cat and dog) is much higher than the similarity between unrelated words (like cat and chair).

Two popular methods to create word embeddings (extending the basic idea) are *word2vec* (presented in [Mikolov et al., 2013]) and *GloVe* (presented in [Pennington et al., 2014]). In this thesis, *GloVe* vectors of size 300 were used.

The word embeddings can be used to create a dense, low-dimension vector representation of a document. This is done by taking the average of all word vectors in the document. In doing so, each sentence is represented by a "centroid word". The efficiency of this method for several tasks has been demonstrated in [Wieting et al., 2015].

2.4.3. Sentence Embeddings and InferSent

Bag-of-words and mean word embeddings lose sequence information. However, the sequence of words in a sentence is important for the meaning. For example, the sentence "*I like cats, not dogs*" has a different meaning than "*I like dogs, not cats*", but both sentences will get the same bag-of-words and mean word embedding vector.

Sentence embeddings aim to learn embeddings for pieces of text instead of single words. In this way, sequence information is taken into account. Several methods have

been proposed to create sentence embeddings (for instance, FastSent [Hill et al., 2016] and SkipTought [Kiros et al., 2015]). In this thesis, *InferSent* as presented in [Conneau et al., 2017] is used.

InferSent learns sentence embedding in a similar way as word embeddings are learned. A neural network is trained on the *Stanford Natural Language Inference (SNLI)* data set ([Bowman et al., 2015]). SNLI contains 570,000 English sentence pairs. Each pair is labelled as *entailment*, *contradiction* or *neutral*. Some examples are presented in Table 2.4.1. The authors assumed that the "semantic nature" makes the data set suitable for learning universal sentence embeddings.

Table 2.4.1.: Example sentences from the *Stanford Natural Language Inference (SNLI)* data set. Taken from `https://nlp.stanford.edu/projects/snli/` (checked: 14.05.2018)

Premise	Hypothesis	Label
A soccer game with multiple males playing.	Some men are playing a sport.	Entailment
A man inspects the uniform of a figure in some East Asian country.	The man is sleeping	Contradiction
A smiling costumed woman is holding an umbrella.	A happy woman in a fairy costume holds an umbrella.	Neutral

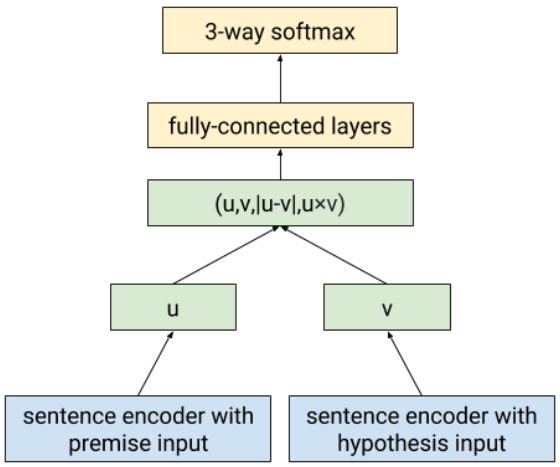


Figure 2.4.1.: Generic training scheme of *InferSent*. Six *sentence encoder* variants were tested. (Adapted from [Conneau et al., 2017])

The architecture of the neural network is presented in Figure 2.4.1. Two separate encoders are used to encode the text and the hypothesis. The embeddings u and v are combined into a feature vector which contains the concatenation, the element-wise product and the element-wise difference of u and v. This vector, containing information from both sentences, is then fed into a fully connected layer and a softmax layer which outputs the probabilities of each label.

The authors tested six architectures for the sentence encoder: LSTM ([Hochreiter and

Schmidhuber, 1997]), GRU ([Cho et al., 2014]), BiLSTM with mean/max pooling ([Collobert and Weston, 2008]), self-attentive networks ([Liu et al., 2016], [Lin et al., 2017]) and hierarchical ConvNet ([Zhao et al., 2015]).

Twelve transfer tasks[17] were used to test the quality of the embeddings. For each task and encoder architecture, the embeddings were used as features. Logistic regression was used as the classification algorithm.

The embeddings generated by the BiLSTM with max pooling and an embedding size of 4096 yielded the best accuracy for SNLI and the transfer tasks. Also, the embeddings were better than other state-of-the-art sentence embeddings like SkipThought.

A pretrained model for InferSent is available on GitHub[18]. This model was used in Section 4.2.

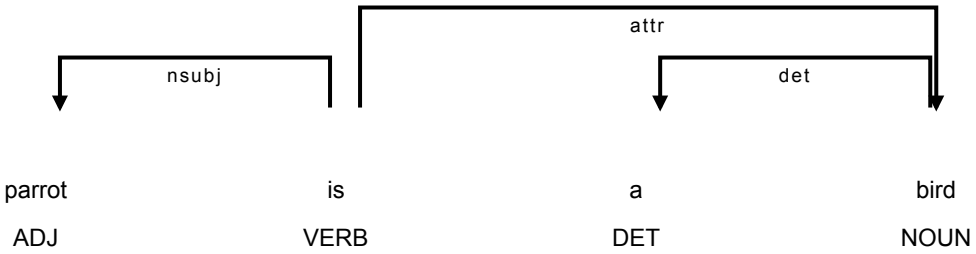
2.4.4. HypeNet and LexNet

HypeNet, presented in [Shwartz et al., 2016], is a method to detect hypernym relations between words. It combines distributional and dependency path based methods to create a vector representation for sentences, which are used as features for the hypernymy detection. LexNet, presented in [Shwartz and Dagan, 2016], is a generalisation of HypeNet, which is able to detect multiple semantic relationships between two words.

The dependency paths add information about joint occurrences of the two words, while the distributional methods add information about the words in separate contexts. Word embeddings are used as distributional features.

For each term pair, all dependency paths were extracted. Each edge was represented as `lemma/POS/dependency label/direction`. An example is given in Figure 2.4.2.

Figure 2.4.2.: Dependency parse of the example sentence *parrot is a bird*, where the relationship between parrot and bird is of interest. This path is represented as `X/NOUN/nsubj/<be/VERB/ROOT/- Y/NOUN/attr/>` (Adapted from [Shwartz et al., 2016]).



The paths were encoded using an LSTM. The average of all encoded paths was used as the path information for the word pair. The combination of distributional and path

[17] For example: binary classification (sentiment analysis, product reviews, subjectivity/objectivity, opinion polarity), multi-class classification (question type), entailment and semantic relatedness, semantic textual similarity, paraphrase detection and caption-image retrieval.

[18] `https://github.com/facebookresearch/InferSent` (checked: 13.05.2018)

information outperformed state-of-the-art techniques.

The task at hand is different from the detection of semantic relations. The relation of two nouns[19] with respect to hypernyms is unambiguous. *Bird* is a hypernym for *parrot* in all cases. Yet the task is not to search for hypernyms or any other semantic relation. It is not possible to assign the correct class by only looking at the objects. For example, there might be sentences saying that *Python* is better than *Ruby*, while others say it is not.

However, it is expected that the dependency path between the objects of interest adds valuable information. In this thesis, the idea on how to extract and encode paths between two words is reused.

[19] The situation might be different if proper nouns are used as well. *Fruit* is an hypernym for *apple*, but not for the company *Apple*.

3. Building a Data Set for Comparative Argument Mining

Due to the novelty of argument mining (and especially comparative argument mining), the supply of data sets is small. Thus, a new data set had to be created.

The data set was designed to answer two questions. First, does a sentence compare two known objects? And secondly, if it does, is the first-mentioned object better or worse than the second-mentioned, according to the sentence context?

The data set was created with data from CommonCrawl[1] and labelled using the crowdsourcing platform CrowdFlower[2]. As described in detail in the following chapters, the annotators were asked to assign one of four (and later three) classes to a sentence in which the objects of interest were highlighted.

The final data set has 7199 sentences, each containing one of 271 distinct object pairs. Each sentence was annotated by at least five annotators.

3.1. Common Crawl Text Corpus

The sentences for the crowdsourcing task were obtained from a CommonCrawl data set. CommonCrawl is a non-profit organisation which crawls the web and releases the crawled data for free use.

The data used in this thesis was already preprocessed (see [Panchenko et al., 2018]). The data contains only English text. Duplicates and near-duplicates were removed, as well as all HTML tags. The texts were split into sentences.

To make the data set manageable, an Elasticearch[3] index was created. The index contains 3,288,963,864 unique sentences.

To get an idea if enough comparative sentences are in the index, it was queried for all sentences containing one of the words "*better*" or "*worse*", as those words often indicate a comparison. This query returns 32,946,247 matching sentences. Querying for "*is better than*" returns 428,932 sentences. Those numbers show that there are enough sentences in the index to create a data set for the given task.

[1] `https://commoncrawl.org` (checked: 23.02.2018)
[2] `https://www.crowdflower.com` (checked: 23.02.2018), now called *Figure Eight*
[3] Elasticsearch is a open-source search engine. `https://www.elastic.co/` (checked: 21.05.2018)

3.2. Prestudies

In preparation to the main crowdsourcing task, several questions had to be answered:

1. How to extract sentences from the index? (How should the query look like?)
2. How to preprocess those sentences? (How to highlight the objects of interest?)
3. Which classes should be assigned to the sentences?
4. How to phrase the guidelines?

Two prestudies were conducted to answer those questions.

3.2.1. First Prestudy: Sentence Sampling and Guidelines

The first prestudy had two goals. First, it should assess if the sentence selection method returns enough comparative sentences. Secondly, the design of the study as described below had to be validated. On that account, a crowdsourcing task with 100 sentences was conducted.

The sentences for the crowdsourcing task must have a high probability of being comparative so that enough positive examples for the machine learning part are present. To ensure this, a list of cue words which indicate comparison was compiled by hand. For the prestudies, these words were "*better*", "*worse*", "*inferior*", "*superior*" and "*because*".

Comparable objects were needed as well. A list of object pairs was selected by hand (see Table 3.2.1). The pairs were selected in a way that they span a wide range of different domains, such as programming languages, countries and pets. The idea behind this is that pets may be compared differently than programming languages. In this way, there will be different comparison patterns in the data.

Table 3.2.1.: Object pairs for the annotation prestudies. The index was queried for sentences which contain both objects and a cue word (*better*, *worse*, *superior*, *inferior* or *because*), to generate sentences for the prestudies.

First Object	Second Object	# Sentences
Android	iPhone	100
beef	chicken	100
BMW	Mercedes	100
cat	dog	100
car	bicycle	100
football	baseball	100
Ruby	Python	100
summer	winter	100
USA	Europe	100
wine	beer	100

Not all comparisons will contain one of the cue words mentioned above. Two different queries were used to overcome the coverage problem. 750 sentences were obtained using

Listing 3.1 (75 for each pair) and 250 using Listing 3.2 (25 for each pair). The second query will also match not-anticipated sentences such as *"I like X more than Y since Z."*.

Listing 3.1: The first query used to extract the sentences for the prestudies from the Elasticsearch index. OBJECT_A and OBJECT_B are placeholders for the first and second object.

```
{
  "query":{
    "bool":{
      "must":[
        {
          "query_string":{
            "default_field":"text",
            "query":"(better OR worse OR superior OR inferior OR
                ↪ because) AND \"<OBJECT_A>\" AND \"<OBJECT_B>\""
          }
        }
      ]
    }
  }
}
```

Listing 3.2: The second query for the prestudies (shortened). This query does not search for the cue words.

```
[...]
          "query_string":{
            "default_field":"text",
            "query":" \"<OBJECT_A>\" AND \"<OBJECT_B>\""
[...]
```

Table 3.2.2 shows some sentences obtained with this method. The objects of interest are printed in italics.

Table 3.2.2.: Sample sentences extracted with the queries from listings 3.1 and 3.2.

Sentence	Cue Words Used
He's the best pet that you can get, Better than a *dog* or *cat*.	Yes
Android phones have better processing power than *iPhone*	Yes
10 Things *Android* Does Better Than *iPhone* OS	Yes
Dog scared of *cat*	No
In fact, many 'supercars' will use *BMW* or *Mercedes* engines.	No

For each sentence, the annotators were asked which of the classes describes the sentence best (see Figure 3.2.1 and Table 3.2.3). The classes `BETTER`, `WORSE` and `NO_COMP` directly refer to the questions stated at the beginning of this chapter. The class `UNCLEAR` was added to capture all sentences which are comparative but do not fit into the classes `BETTER` or `WORSE`. For instance, if both objects of interest appear in the sentence, but they

are not compared against each other.

Figure 3.2.1.: An example for questions presented to the annotators in the first prestudy.

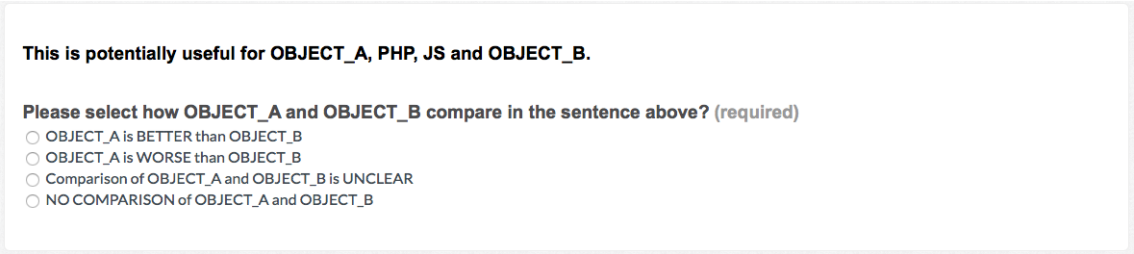


Table 3.2.3.: The classes for the first and second prestudy. They correspond to the answers the annotator could select in Figure 3.2.1

Class	Description
`BETTER`	The first object in the sentence (`OBJECT_A`) is better than the second one (`OBJECT_B`)
`WORSE`	The first object is worse
`UNCLEAR`	Neither `BETTER` nor `WORSE` fits, but the sentence is comparative
`NO_COMP`	The sentence is not comparative or the sentence is a question

In each sentence, the first object of interest was replaced with `OBJECT_A`, while the second one was replaced with `OBJECT_B` (see Table 3.2.4). The idea behind this was to enable the annotators to quickly see which objects should be taken into account for assigning a class. Also, they should not be biased by personal preference. For example, in sentence two of Table 3.2.4, the annotator might be confused which of the objects are of interest, yet the replacement makes it clear that he should ignore "*C#*" and "*VB*".

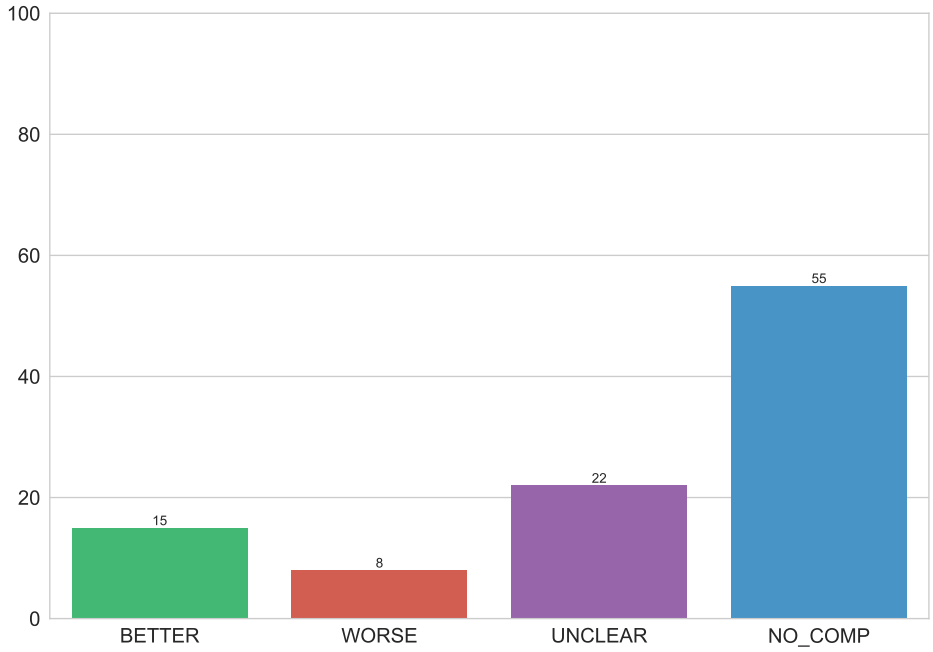
Each annotator saw five sentences to annotate per page. He was also able to look into the annotation guidelines anytime he wanted. To filter out low-quality annotators, twelve sentences were selected as test questions. Each participant took a quiz with eight test questions before the actual annotation process. One out of five sentences was a test question as well. The annotator had to keep an accuracy of 70% on the test questions, otherwise he was removed from the task.

Figure 3.2.2 shows the class distribution of the annotation results. As 45 sentences are comparative, the selection procedure worked satisfactory.

Table 3.2.4.: Examples of uncertain sentences for the *first prestudy*. The annotators could not agree on one class for this sentences.

#	Sentence	BETTER	WORSE	UNCLEAR	NO_COMP
1	While *OBJECT_A* is slightly faster, *OBJECT_B* utilises memory better	1	1	1	0
2	Your C# and VB devs can suddenly easily write web apps and your *OBJECT_A* and *OBJECT_B* devs can too - with the added bonus of much better performance.	0	0	2	2
3	The only reason *OBJECT_A* is used over *OBJECT_B*, is because of libraries..	0	1	1	1
4	for json: i also think its better to just use *OBJECT_A*, *OBJECT_B*, perl and transform it	1	0	1	1

Figure 3.2.2.: The results of the first prestudy. As 45 sentences are comparative, the selection procedure worked satisfactory.



The confidence of the annotations was acceptable (see Table 3.2.5). Only for eight sentences, no class got the majority of votes. However, only 37 sentences got a clear result.

Table 3.2.5.: Annotation confidence for the *first prestudy*. The confidence is calculated as *judgments for majority class / total judgments*. Less than 51 percent confidence means no class got the majority of votes, while 100 percent means that all annotators voted for the same class.

Confidence	Sentences	% of data set
100%	37	37
91-99%	0	0
81-90%	0	0
71-80%	5	5
61-70%	50	50
51-60%	0	0
0-50%	8	8

Some uncertain sentences are shown in Table 3.2.4 together with the number of decisions per class. As one can see in sentence two, annotators often were not able to distinguish between `NO_COMP` and `UNCLEAR`. This is true for the majority of cases were more than one class was assigned.

Fourteen out of 55 participants took part in an exit survey to rate the task. The overall satisfaction was rated with 3.2 out of 5. While the instructions (4.5), difficulty (4.4) and payment (3.8) got acceptable to good ratings, the test questions (2.9) were criticised. Also, 32 of 85 potential annotators failed the quiz. A second prestudy was conducted to address the discovered problems.

3.2.2. Second Prestudy: Sentence Preprocessing and Rephrasing of the Guidelines

Two-hundred sentences were annotated in the second prestudy. Some aspects were identical to the first prestudy. As the sentence selection process worked fine, the same 1000 base sentences were used in the second prestudy. Each sentence was annotated by three annotators. The annotators saw five sentences per page, one being a test question. They had to pass a quiz of eight test questions and had to keep an accuracy of 70% on the test questions during the annotation procedure. The classes were the same as in the first prestudy (see Table 3.2.3).

To address the shortcomings mentioned above, the task design was changed in several aspects. As the pair "*Ruby vs. Python*" requires knowledge in computer science, the need was expressed in the title of the task. To address the problem with the confusion between `UNCLEAR` and `NO_COMP`, the wording on this classes was changed (see Figure 3.2.3).

Some annotators of the first prestudy complained that the test questions were not fair. In fact, they contained some special cases so that did not represent the whole data set appropriately. In the second prestudy, more (51 instead of 12) test questions were used. The

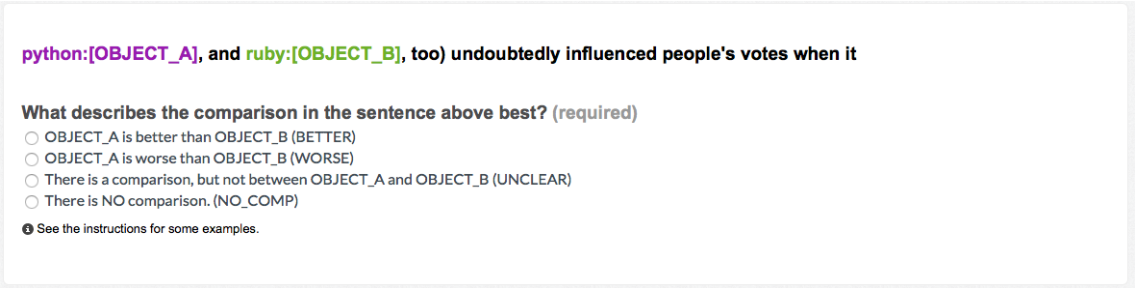


Figure 3.2.3.: The new annotator view for the second prestudy. The objects are highlighted by colour.

new test questions covered easy and hard cases. Explanations for the harder test questions were added. The annotator saw those explanations after he failed the test question.

The sentence preprocessing was adjusted. Instead of replacing the object, **:[OBJECT_A]** or **:[OBJECT_B]** was appended. The colon and square brackets emphasise where the object of interest ends and the suffix begins. The idea behind this was that the removal of the objects also removed some context from the sentences, which might be useful to classify them correctly. In addition, the objects were shown in a different colour than the rest of the text.

The class distribution of the 200 sentences is presented in Figure 3.2.4. As in the first prestudy, a sufficient amount of the sentences are comparative. The confidence of the annotations (see Table 3.2.6) was better than in the first prestudy. However, the confusion between `UNCLEAR` and `NO_COMP` is still the main problem. Compared to the first prestudy, the amount of sentences where all annotators agreed on one class increased from 37% to 62%. Table 3.2.7 shows some of the uncertain sentences.

Table 3.2.6.: Annotation confidence for the second prestudy. The confidence is calculated as *judgments for majority class / total judgments*.

Confidence	Sentences	% of data set
100%	124	62.00
91-99%	2	1.00
81-90%	2	1.00
71-80%	5	2.50
61-70%	57	28.50
51-60%	2	1.00
0-50%	8	4.00

Figure 3.2.4.: The results of the second prestudy.

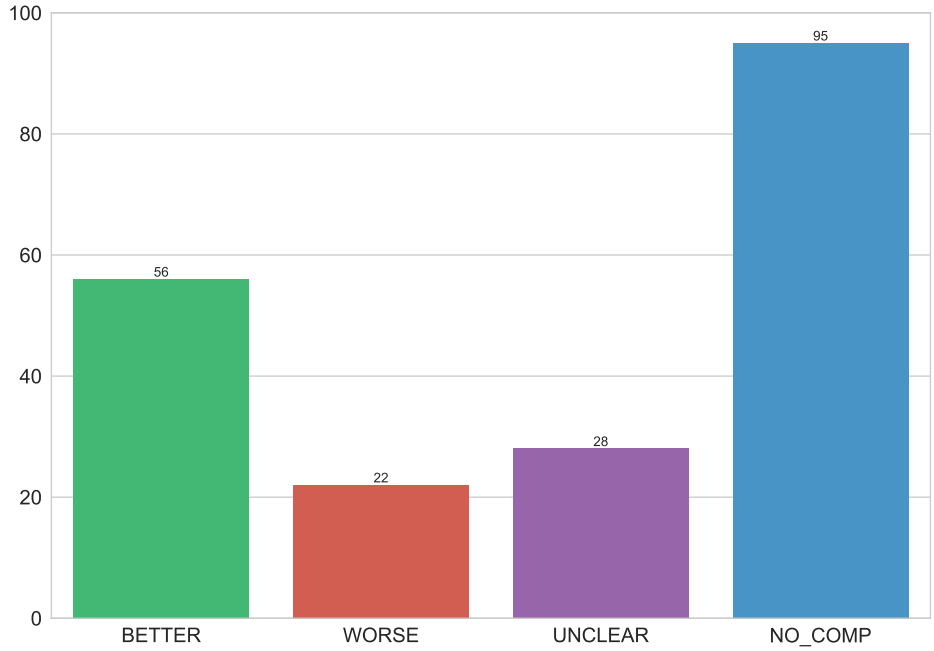


Table 3.2.7.: Examples of uncertain sentences for the second prestudy. The annotators could not agree on one class for these sentences.

Sentence	BETTER	WORSE	UNCLEAR	NO_COMP
Ruby:[OBJECT_A] and **Python:[OBJECT_B]** perform significantly better.	0	0	2	2
Unfortunately, when it comes to potential projects **Ruby:[OBJECT_A]** suffers because it's similarity to **Python:[OBJECT_B]**	1	2	1	0
Not to mention that the **iPhone:[OBJECT_A]** and **Android:[OBJECT_B]** phones deliver a far superior user experience overall	1	0	1	1
Google shouldn't have mandated an inferior map app on the **iPhone:[OBJECT_A]** (as opposed to **Android:[OBJECT_B]**).	1	1	0	1

Thirty-five out of 125 annotator candidates failed the initial quiz. Twelve annotators were removed during the annotation process as they answered too many test questions wrong.

Twenty-two annotators took the exit survey. The overall satisfaction increased to 3.7 out of 5. The test question fairness was now rated with 3.7 instead of 2.9. The rating for the payment slightly increased to 3.9, yet the payment was not changed. Furthermore, the rating for the instructions decreased to 3.9 and for the difficulty to 3.5. The change in numbers can be explained by the increased amount of sentences, which introduce new cases which are not directly reflected in the annotation guidelines.

3.2.3. Discussion

Only a small fraction of the annotators took the exit surveys in both prestudies. Still, they gave valuable hints to improve the study design. Yet, the increased confidence of the annotations is the more important signal.

Using two comparable objects and cue words to query the index returns a satisfying amount of comparative sentences.

The changes in the second prestudy were well received by the annotators. In the end, they improved the quality of results as there are more cases where all annotators agreed on one class.

The distinction between `UNCLEAR` and `NO_COMP` was an issue. This illustrates that the choice of a class is subjective to some degree.

Due to an error in the creation of the crowdsourcing task, the sentences were not shuffled. This means that the first 100 sentences of the second prestudy were the same as the 100 sentences of the first prestudy. Another problem was the bias: all sentences contained only the pairs *Ruby vs. Python* and *Android vs. iPhone*. Because the goal of the prestudy was mainly to assess the sentence selection method and the guidelines, this does not invalidate the results. These problems were removed in the main study.

All in all, the prestudy was successful. There are only few cases where no agreement on the class could be achieved. The prestudy showed that the task at hand is not easy, but feasible.

3.3. Main Study

The insights from the prestudies influenced the design of the main study.

The definition of classes changed (see Table 3.3.1). UNCLEAR was renamed to OTHER, NO_COMP to NONE. Those names are a better description for the classes. Eventually, after the first 2250 sentences were finished the class OTHER was dropped completely (see Section 3.3.2). The change was reflected in the annotation guidelines as well.

Table 3.3.1.: The final classes for the main study.

Class	Description
BETTER	The first object in the sentence is better than the second object
WORSE	The first object is worse
NONE	Neither better nor worse fit

Instead of one big task, several tasks per domain were created. All tasks used the same annotation guidelines. Each sentence was at least annotated by five annotators.

3.3.1. Sentence Sampling Method and Domain Selection

The sentence selection process was similar to the prestudy. The pairs and the cue words (see Figure 3.3.1) changed. The cue words were generated using *JoBimText*[4], a software package for distributional semantics. JoBimText was queried for the words most similar to *better* and *worse*, so that more comparisons were captured by the selection process.

Figure 3.3.1.: It is expected that these cue words appear frequently in comparative sentences, which makes them suitable keywords for queries on the index.

better	decent	harder	nastier
easier	safer	slower	inferior
faster	superior	poorly	mediocre
nicer	solid	uglier	
wiser	terrific	poorer	
cooler	worse	lousy	

Three domains were selected for the object pairs. The domains were chosen in a way that a majority of people might be able to decide whether a sentence contain a comparison or not. Also, a wide range of comparison patterns should be included in the data.

The most specific domain was "*computer science concepts*". It contained objects like programming languages, database products and technology standards such as Bluetooth and Ethernet. Many computer science concepts can be compared objectively. For instance, one can compare Bluetooth and Ethernet on their transmission speed. Some basic knowledge of computer science was needed to label sentences correctly: to compare

[4] http://ltmaggie.informatik.uni-hamburg.de/jobimviz/ (checked: 28.02.2018)

Eclipse and NetBeans, the annotator must know what an Integrated Development Environment (IDE) is and that both objects are Java IDEs. The need for this knowledge was communicated to the prospective annotators. The objects for this domain were manually extracted from "*List of ...*" articles from Wikipedia.

The second, broader domain was "*brands*". It contains objects of different types (for instance car brands, electronics brands, and food brands). As brands are present in everyday life of people, it is expected that anyone can label the majority of sentences containing well known brands such as "*Coca-Cola*" or "*Mercedes*". As with computer science, the objects for this domain were extracted from "*List of ...*" articles from Wikipedia.

The last domain was not restricted to any topic. For each one of 24 randomly selected seed words, ten similar words were extracted using JoBimText. The seed words (see Figure 3.3.2) were created using `randomlists.com`[5]. Listing 3.3 shows the result[6] for the seed word "*Yale*". Duplicates or too broad terms (like *university*) were removed manually.

Figure 3.3.2.: These seed words were used to create pseudo-random pairs for the *Random* domain.

book	coffee	Hoover	pencil	Tolkien
car	cork	Metallica	salad	wine
carpenter	Florida	NBC	soccer	wood
cellphone	hamster	Netflix	Starbucks	XBox
christmas	hiking	ninja	sword	Yale

Listing 3.3: The JoBimText response for *"Yale"*. These objects can be compared in a meaningful way.

```
[...]
   "results":
      [{"score":701.0,"key":"yale#NP"},
      {"score":245.0,"key":"harvard#NP"},
      {"score":151.0,"key":"princeton#NP"},
      {"score":135.0,"key":"mit#NP"},
      {"score":135.0,"key":"cornell#NP"},
      {"score":121.0,"key":"stanford#NP"},
      {"score":116.0,"key":"university#NP"},
      {"score":111.0,"key":"nyu#NP"},
      {"score":111.0,"key":"university#NN"},
      {"score":109.0,"key":"dartmouth#NP"}]
```

In the following, this domain is called *random*. Some example pairs for all domains are

[5] `https://randomlists.com` (checked: 25.01.2018)

[6] `http://ltmaggie.informatik.uni-hamburg.de/jobimviz/ws/api/stanford/jo/similar/harvard%23NP?numberOfEntries=10&format=json` (checked 25.01.2018); Some unimportant fields were removed for brevity

shown in Table 3.3.2.

Table 3.3.2.: Example object pairs for the main study. These pairs were used to extract sentences from the Elasticsearch index.

Brands	Computer Science	Random
Microsoft vs. Apple	Java vs. Python	baseball vs. hockey
Nikon vs. Leica	Eclipse vs. Netbeans	fishing vs. swimming
Coca-Cola vs. Pepsi	OpenGL vs. Direct3D	SUV vs. minivan
Nike vs. Adidas	Integer vs. Float	Kennedy vs. Nixon
Ibuprofen vs. Advil	USB vs. Bluetooth	plastic vs. wood
Ford vs. Honda	Oracle vs. MysQL	Harvard vs. Princeton

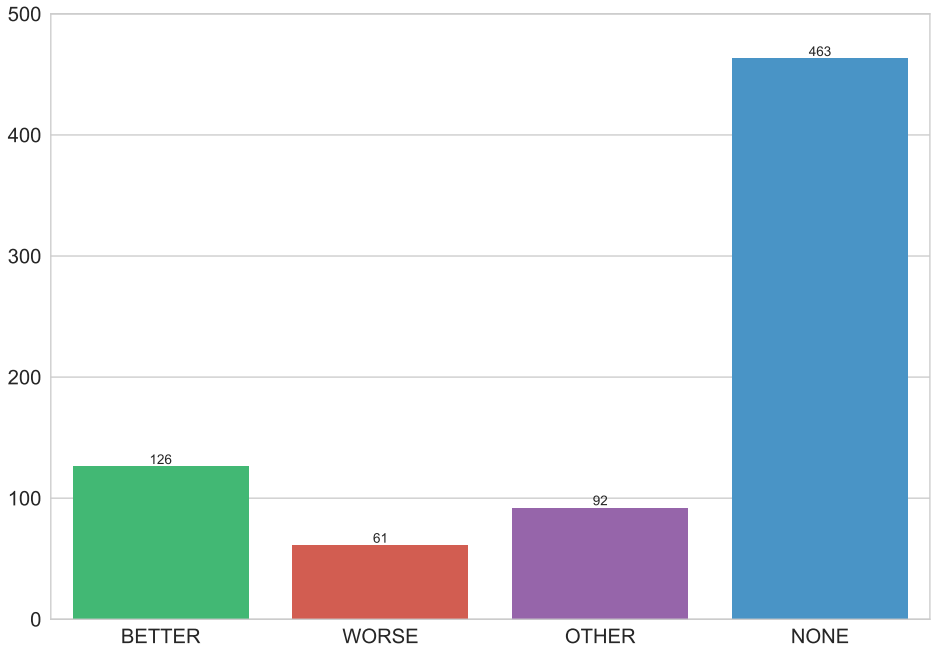
Especially for the domains brands and computer science, the object lists were long (4493 brands and 1339 for computer science). The frequency of each object was checked using a frequency dictionary to reduce the number of possible pairs. All objects with a frequency of zero and ambiguous objects were removed from the list. For instance, the objects "*RAID*" (a hardware concept) and "*Unity*" (a game engine) were removed from the computer science list as they are also regularly used nouns.

The remaining objects were combined to pairs. For each object type, all possible combinations were created. For brands and computer science, the type was the URL of the Wikipedia page. For the random domain, the seed word was used. This procedure guaranteed that only meaningful pairs were created.

The index was queried for entries containing both objects of each pair. For 90% of the queries, the cue words were added to the query. All pairs were the query yielded at least 100 sentences were kept.

From all sentences of those pairs, 2500 for each category were randomly sampled as examples for the crowdsourcing task. To check the sentence selection method once again, a small, random subset of the sentences was labelled by the author prior to the crowdsourcing task. These labels were discarded for the crowdsourcing task. The label distribution of the 742 sentences is presented in the Figure 3.3.3. As the numbers are similar to the prestudy, the procedure is still valid.

Figure 3.3.3.: Result of 742 manually labelled sentences. The labelling was done to see what result could be expected from the crowdsourcing task and if the sentence sampling method still works.



3.3.2. Domain Subset: Brands

For the Brands domain, 2335 sentences were annotated. The sentences contained objects of 36 pairs. As shown in Table 3.3.3, the annotators could agree on one class for the majority of the sentences.

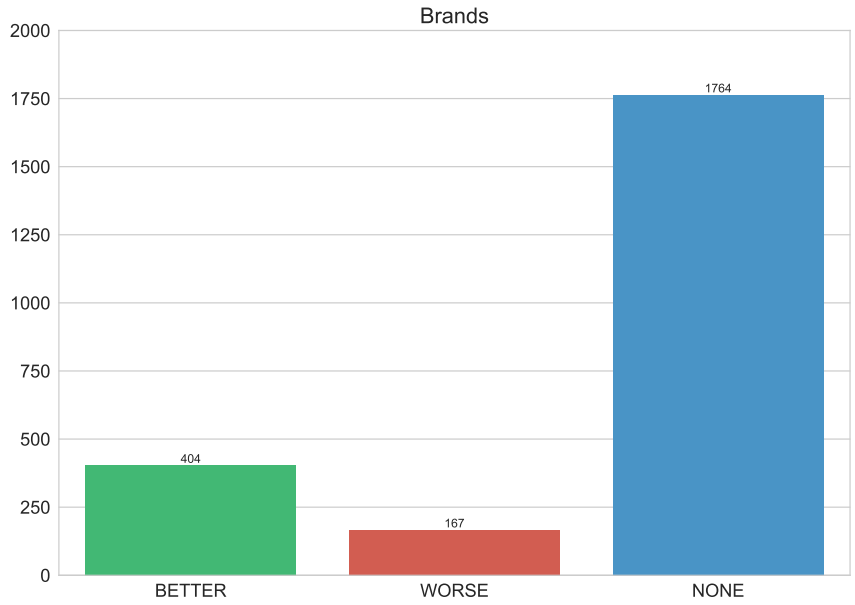
Table 3.3.3.: Annotation confidence for the domain *Brands*. The confidence is calculated as *judgments for majority class / total judgments*.

Confidence	Sentences	% of data set
100%	1719	71.30
91-99%	0	0.00
81-90%	34	1.41
71-80%	337	13.98
61-70%	8	0.33
51-60%	256	10.62
0-50%	57	2.36

The class distribution is presented in Figure 3.3.4. The amount of comparative sentences is lower (24.45%) than in the prestudy. The reason for this is seen in the abandonment of the `OTHER` (`UNCLEAR`) class.

Even with the renaming of `UNCLEAR` to `OTHER` and the rephrasing of the descrip-

Figure 3.3.4.: Class distribution for sentences of the domain *Brands*



tion, the class was too confusing for the annotators. The class was too similar to NONE (NO_COMP). Eventually, all sentences labelled as OTHER were merged into NONE. This decision was made after 750 sentences were labelled for each domain. First machine learning experiments also showed that OTHER is not distinguishable from NONE for all tested features and algorithms.

For the first 750 sentences (with four classes), 47% failed the initial quiz (357 out of 755). During the annotation process, 13% answered to many test questions wrong. The numbers improved after the class OTHER was removed: only 25% failed the initial quiz and 8% were removed during the annotation process.

150 annotators took the exit survey to rate the task on a scale from one to five. Overall, the task was rated with 3.7. The instructions got a rating of 4.0, the fairness of test questions 3.5, the difficulty 3.6 and the payment 3.7.[7] Broken down, the four-class task (67 participants) got a rating of 3.5, the three-class task (83 participants) 3.8.

3.3.3. Domain Subset: Computer Science

For the Computer Science domain, 2425 sentences (containing one of 42 pairs) were annotated. The confidence of the annotations (Table 3.3.4) is satisfactory. The class distribution (Figure 3.3.5) is better, as more sentences are comparative.

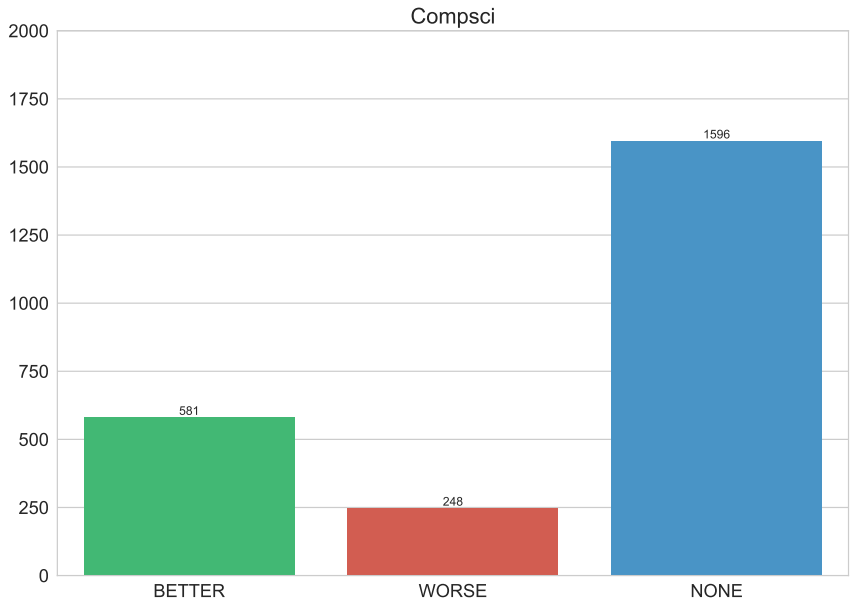
As with the domain Brands, 48% failed the initial quiz for the task with four classes.

[7] The numbers are the average of the survey results for each task created on CrowdFlower, weighted by the number of participants.

Table 3.3.4.: Annotation confidence for the domain *Computer Science*. The confidence is calculated as *judgments for majority class / total judgments*.

Confidence	Sentences	% of data set
100%	1698	70.02
91-99%	0	0.00
81-90%	25	1.03
71-80%	369	15.22
61-70%	18	0.74
51-60%	246	10.14
0-50%	69	2.85

Figure 3.3.5.: Class distribution for sentences of the domain *Computer Science*



25% dropped out during the annotation process. Again, the numbers improved after `OTHER` was removed. Only 15% failed the quiz and 7% were removed during the annotation process.

One hundred and six annotators took the exit survey. The task was rated with 3.9. The instructions[8] got a rating of 4.2, test question fairness 4.1, difficulty 3.9 and payment[9] 3.9 as well. The four-class task (27 participants) got a rating of 3.5, the three-class task (79 participants) 3.9.

[8] The instructions were the same for all tasks.
[9] All tasks had the same payment.

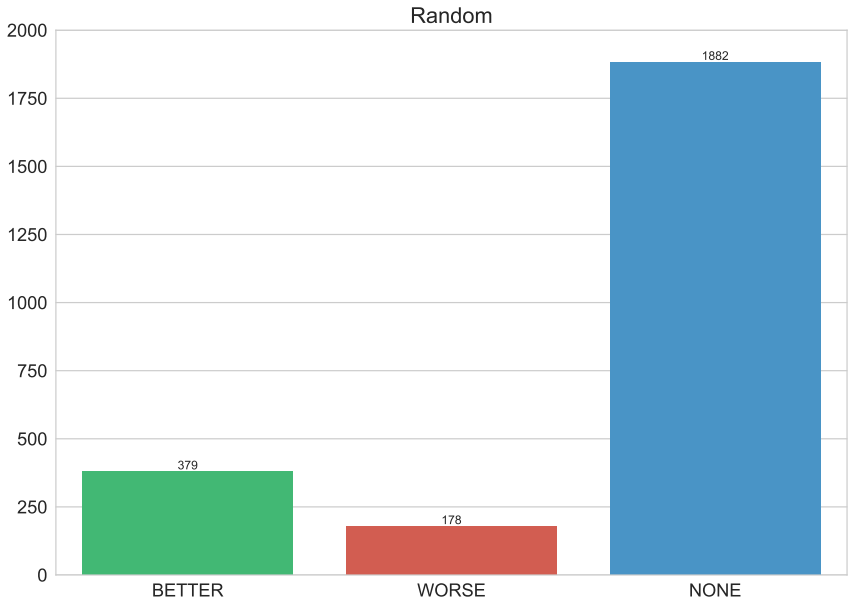
3.3.4. Domain Subset: Random

The Random domain contains 2439 sentences with 167 pairs. The confidence values (Table 3.3.5) and class distribution (Figure 3.3.6) are satisfactory.

Table 3.3.5.: Annotation confidence for the domain *Random*. The confidence is calculated as *judgments for majority class / total judgments*.

Confidence	Sentences	% of data set
100%	1753	71.87
91-99%	0	0.00
81-90%	16	0.66
71-80%	362	14.84
61-70%	7	0.29
51-60%	252	10.33
0-50%	49	2.01

Figure 3.3.6.: Class distribution for sentences of the domain *Random*



As with the other domains, the first 750 sentences (with `OTHER`) performed worse than the rest: 44% dropped out during the quiz, 16% were removed during the annotation process. For the three-class scenario, the numbers improved to 14% (quiz) and 7% (annotation task).

Ninety-seven participants rated the task with 3.6 (four classes: 3.1, 29 participants; three classes: 3.9, 68 participants). The instructions got a rating of 4.0, test question fairness 3.8, difficulty of the task 3.8 and payment 3.5.

3.3.5. Discussion

Table 3.3.6 summarises the confidence of the annotations on all three domains. The annotators could agree on one class for the majority of the sentences. Just for 169 sentences (2.35%), no class got the majority of votes.

Table 3.3.6.: Annotation confidence for all domains. The confidence is calculated as *judgments for majority class / total judgments*.

Confidence	Sentences	% of data set
100%	5111	71.00
91-99%	0	0.00
81-90%	75	1.04
71-80%	1057	14.68
61-70%	33	0.46
51-60%	754	10.47
0-50%	169	2.35

Table 3.3.7 shows some examples on uncertain sentences. The first two sentences are comparative. Knowledge is needed to label the first sentence correct. A decision for the second sentence is hard because the context is missing. For instance, it depends on the use case if the hardness of stone is better or worse than the hardness of metal.

It is unclear if *Groovy* and *Java* are compared in sentence three. On one hand, one can understand this sentence in a way *Groovy* is easier than *Java*. On the other hand, it can be understood in a way that *Groovy* supports Java programmers.

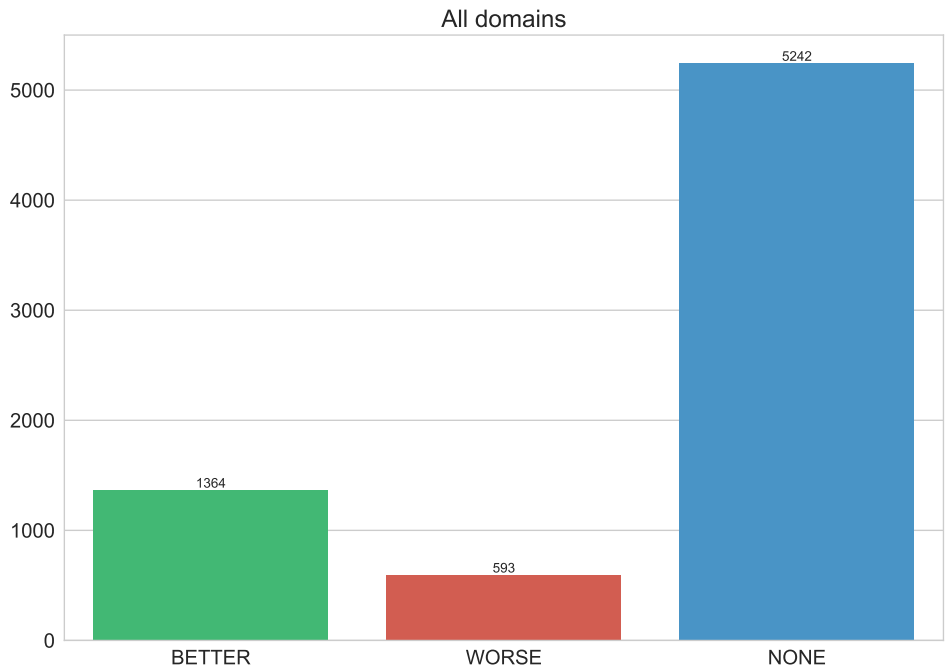
The fourth sentence does not explicitly states that one is better than the other.

Table 3.3.7.: Examples of uncertain sentences of the main study. The annotators could not agree on one class for these sentences.

#	Sentence	# BETTER	# WORSE	# NONE
1	Goodnight **NetBeans:[OBJECT_A]**, Hello **Eclipse:[OBJECT_B]**	2	2	1
2	**stone:[OBJECT_A]** is harder than **metal:[OBJECT_B]**.	1	2	2
3	The new version of the **Groovy:[OBJECT_A]** programming language aims to make life easier for programmers who work with **Java:[OBJECT_B]** and SQL, the language's developers note	3	0	3
4	Even if this **juice:[OBJECT_A]** isn't your typical **cider:[OBJECT_B]**, it's just as good if not better in our opinion!	2	1	2
5	Only Nevada (14.4 percent), **michigan:[OBJECT_A]** (13 percent) and **california:[OBJECT_B]** (12.4 percent) were worse.	2	1	2

The class distribution (Figure 3.3.7) is similar to the prestudies. As expected, the majority of sentences is not comparative. The class `BETTER` is more than twice as big as the class `WORSE`, which is expected to complicate the classification.

Figure 3.3.7.: Distribution of classes in the final crowdsourcing data set.



In the end, the crowdsourcing task was successful. For the majority of sentences (71.00%) the annotators could agree on one class, while the amount of unclear sentences is small (2.35%).

4. Classification of Comparative Sentences

The data collected from the crowdsourcing task was used as training data for two classification problems. In the first problem, a machine learning algorithm was trained to predict one of the three classes per sentence (see Table 3.3.1). The second problem is a simplification of the first one as it is designed as a binary classification problem. The classes `BETTER` and `WORSE` were merged into the class `ARG`.

The data was split into a training set (5759 sentences; 4194 `NONE`, 1091 `BETTER` and 474 `WORSE`) and a held-out set. The experiments described in this chapter were conducted on the training set only. During the development, the experiments were evaluated using stratified k-fold cross-validation where k equals five.

The held-out set stayed untouched until the final evaluation presented in Section 4.4.1.

If not stated otherwise, scikit-learn (presented in [Pedregosa et al., 2011]) was used to perform feature processing, the classification and evaluation.

4.1. Classification Algorithm Selection

To find the most suitable classification algorithms, thirteen (see Figure 4.1.1) were selected and compared. Except *XGBoost*[1] and *Extra Trees Classifier*, all algorithms were used in at least one paper presented in Section 2.1. A binary bag-of-words model computed on the whole sentence (see Section 4.2) was used as the feature. The f1 score was used as the measure to compare the algorithms.

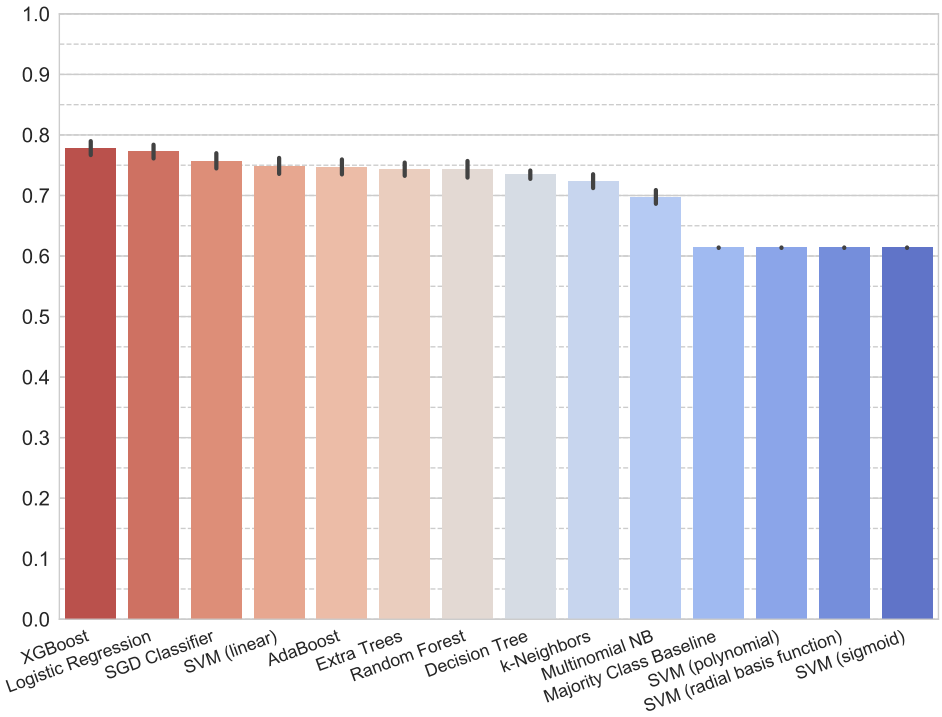
Tree-based methods and linear models worked well. Support Vector Machines with non-linear kernels assigned `NONE` to all sentences.

As XGBoost and Logistic Regression achieved high f1 scores, no further investigations on the performance of other algorithms was done. A set of hyper-parameters for XGBoost was tested using exhaustive grid search and randomized search. However, no significant increase in the f1 score could be achieved.

In the following sections, all experiments were conducted using XGBoost with 1000 estimators.

[1] XGBoost is not part of scikit-learn. The implementation presented in [Chen and Guestrin, 2016] was used.

Figure 4.1.1.: F1 score of all tested classification algorithms. A binary bag-of-words feature was used as the baseline feature. Each algorithm was trained with five stratified folds of the data. The black bars show the standard derivation.



4.2. Features

Several vector representations were tested as features. The simplest one was a binary *bag-of-words* model realised with scikit-learn's `CountVectorizer`. Another vector representation was generated with the five-hundred most frequent part-of-speech bi-, tri- and four-grams (called *POS n-grams*). The *mean word embedding* vector was created by calculating the mean of each word's GloVe vector (as contained in spaCy's `en_core_web_lg`[2] model). The pretrained *InferSent* model[3] was used to create sentence embedding vectors.

A boolean feature capturing the appearance of a comparative adjective (called *Contains JJR*) [4] was tested as well. Part-of-speech tagging for all features was done with spaCy.

Two preprocessing steps were used to generate the input for the feature calculation.

The first preprocessing step decided if the full sentence or a part of it should be used. The *first part* contained all words from the beginning of the sentence to the first object, while the *last part* contained all words from the second object to the end of the sentence. The *middle part* contained all words between the first and the second object.

[2] `https://spacy.io/models/en#section-en_core_web_lg`
[3] `https://github.com/facebookresearch/InferSent` (checked 13.05.2018)
[4] Tag *JJR* in the Penn Treebank (`https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`)

Table 4.2.1.: Preprocessing examples for the sentence "*In my mind, Python is better than Ruby*"

Step 1	Step 2	Result
Middle part	untouched	Python is better than Ruby
Middle part	removal	is better than
Full sentence	distinct replacement	In my mind, OBJECT_A is better than OBJECT_B
First part	removal	In my mind,

The second step was done to check the importance of the objects for the classification. The objects either stayed untouched, were removed or replaced. Two different replacement strategies were tested. First, both objects were replaced by the term *OBJECT* (*replacement*). Second, the first object was replaced by *OBJECT_A* and the second by *OBJECT_B* (*distinct replacement*). This resulted in sixteen versions of each of the features mentioned above (four parts $\times$ four object strategies). Some examples are shown in Table 4.2.1.

Two features based on LexNet were created to encode dependency parsing information. The original code of LexNet was used to create the string representation of paths, as described in Section 2.4.4. An LSTM was used to create path embeddings out of the string paths. Because the paper does not mention any details about the LSTM encoder, different architectures and hyper-parameter values were tested. The best results were achieved with the architecture described in Figure 4.2.1.

The addition of more layers or neurons did not increase the performance of the network. This is also true for adding bidirectionality to the LSTM layer. The paths (encoded as one-hot vectors) were used as targets for the network. Keras'[5] embedding layer was used to create word embeddings of length 100 for the string path components.

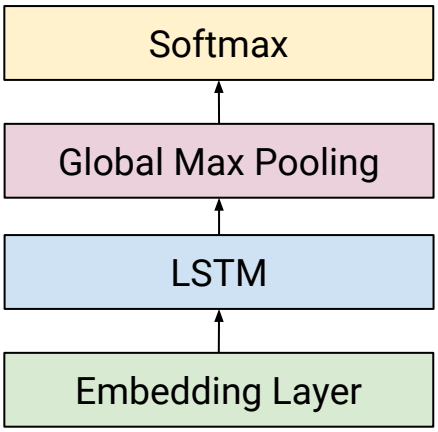
Different setups for the string path creation were tested. In the original implementation, the paths were restricted to a length of four. The directionality of edges was restricted as well. The first object must be reachable from the lowest common head of the two objects by following left edges only, the second one by following right edges. In the following, this setup is called *original*. However, only 1519 sentences from the training set got a path with these restrictions.

To overcome this problem, the restrictions were relaxed. The second LexNet setup (called *customised*) limits the paths to a size of sixteen and abolishes the directionality restriction. With this setup, only 399 sentences did not get a path.[6]

[5] A software package for neural networks. [Chollet et al., 2015]

[6] All sentence without a generated path got the artificial path *NOPATH*. The customised version of LexNet is available at `https://github.com/ablx/LexNET`

Figure 4.2.1.: Architecture of the LSTM path encoder. The embedding layer created word embeddings for the path edges, which were fed into an LSTM with 200 neurons. The result wa pooled (pool size of two) and fed into a softmax layer. Keras was used to implement the network. The network was trained for 150 epochs with a batch size of 128 and RMSprop as the optimizer.



4.3. Classification Experiments

4.3.1. Baselines

As described in Section 2.1, there is no task which is similar enough to the one at hand to use as a baseline. Thus, two baselines were created. The first baseline, shown in Table 4.3.1 and 4.3.3, was created by assigning classes to the data at random, respecting the distribution of classes in the original data.

Table 4.3.1.: Random (stratified) baseline for the three-class scenario.

	precision	recall	f1 score
BETTER	0.19 ±0.01	0.21 ±0.01	0.20 ±0.01
WORSE	0.06 ±0.02	0.05 ±0.02	0.06 ±0.03
NONE	0.73 ±0.00	0.73 ±0.00	0.73 ±0.00
avg.	0.57 ±0.00	0.58 ±0.01	0.57 ±0.00

Table 4.3.2.: Majority class baseline for the for the three-class scenario.

	precision	recall	f1 score
BETTER	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
WORSE	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
NONE	0.73 ±0.00	1.00 ±0.00	0.84 ±0.00
avg.	0.53 ±0.00	0.73 ±0.00	**0.61** ±0.00

The second baseline predicts all sentences as NONE. The results are shown in tables 4.3.2 and 4.3.4.

Table 4.3.3.: Random (stratified) baseline for the binary scenario.

	precision	recall	f1 score
ARG	0.26 ±0.03	0.26 ±0.03	0.26 ±0.03
NONE	0.72 ±0.01	0.72 ±0.01	0.72 ±0.01
avg.	0.60 ±0.02	0.60 ±0.02	0.60 ±0.02

Table 4.3.4.: Majority class baseline for the binary scenario.

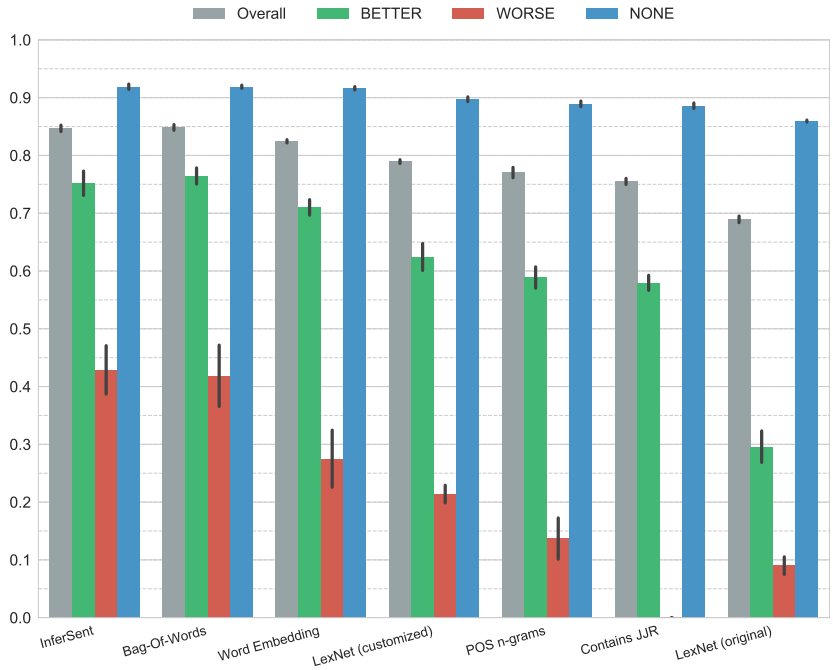
	precision	recall	f1 score
ARG	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
NONE	0.73 ±0.00	1.00 ±0.00	0.84 ±0.00
avg.	0.53 ±0.00	0.73 ±0.00	**0.61** ±0.00

Scikit-learns `DummyClassifer` was used for all baselines.

4.3.2. Results

The classification results of the best performing feature configurations in the three-class scenario are presented in figures 4.3.1 (f1 score), 4.3.2 (precision) and 4.3.3 (recall). Each feature was tested and evaluated using five stratified folds. The black bar shows the standard derivation. All scores were calculated with scikit-learn's metric module. All features except the *LexNet (original)* features used the middle part of the sentence and left the objects untouched. In the LexNet features, the objects were replaced with *Objecta* and *Objectb*. *LexNet (original)* used the full sentence.

Figure 4.3.1.: F1 score for the three-class scenario using XGBoost. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the f1 score on the y-axis. The black bar shows the standard derivation.



The best feature (bag-of-words) yielded a score 24 points above the baseline. The worst feature (*LexNet (original)*) was still eight points above the baseline. Bag-of-words (f1 score 0.848) and InferSent (f1 score 0.842) delivered almost identical results. The boolean feature that captures comparative adjectives in the middle of the sentence yielded a f1 score over the baseline as well. However, it did not assign any examples to the class `WORSE`.

Despite the fact that only 1519 sentences got a path embedding for *LexNet (original)*, the feature is able to predict some sentences correctly. This indicates that this feature setup is reasonable and would work probably work well if more examples were present. An experiment with only the 1519 sentences confirmed this, as the feature was then able to achieve an f1 score of 0.75.

Figure 4.3.2.: Precision for the three-class scenario using XGBoost. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the precision score on the y-axis. The black bar displays the standard derivation.

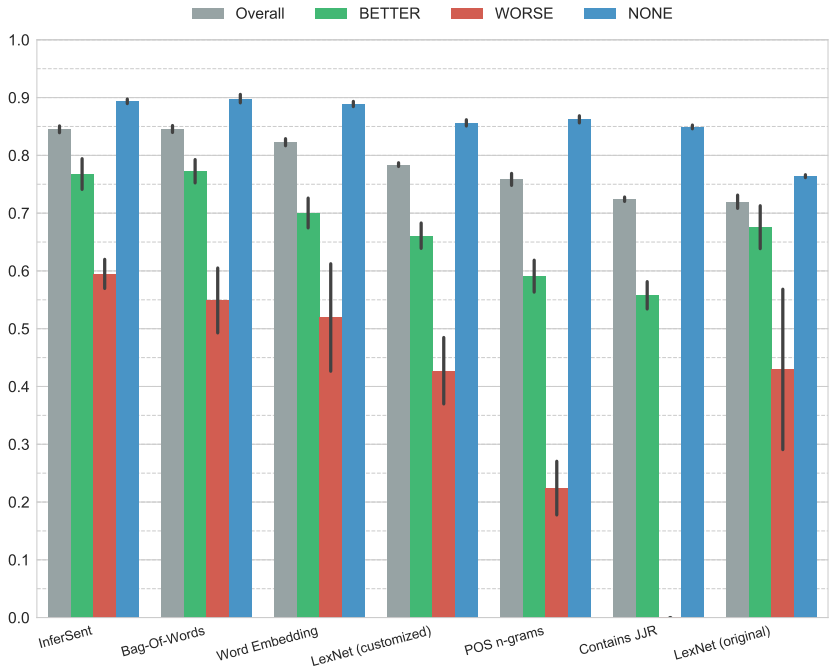
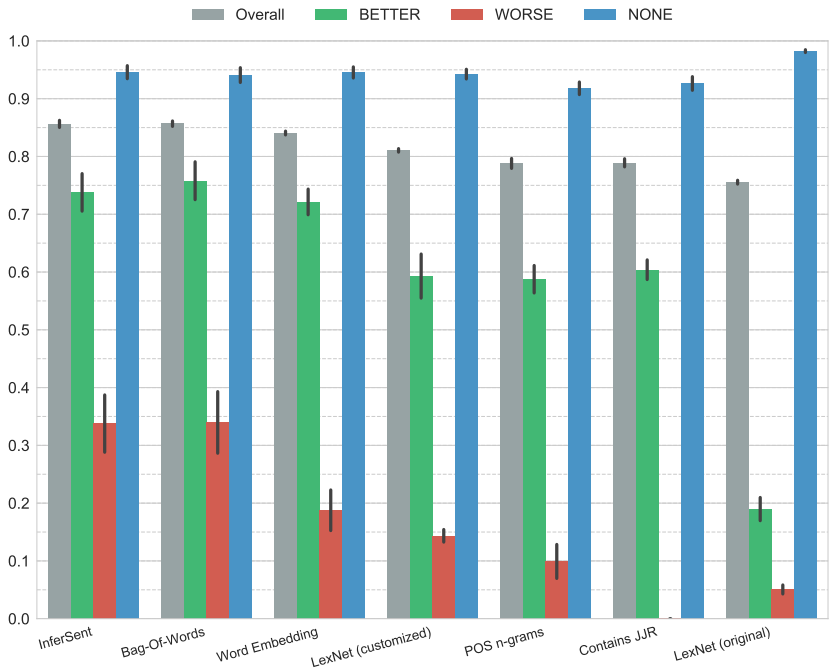


Figure 4.3.3.: Recall for the three-class scenario using XGBoost. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the recall score on the y-axis. The black bar shows the standard derivation.



Figures 4.3.4 (f1 score), 4.3.5 (precision) and 4.3.6 (recall) show the results for the binary classification.

As with the three-class scenario, InferSent (f1 score 0.886) and the bag-of-words (f1 score 0.882) performed best and achieved almost equal results. They are closely followed by mean word embeddings. In summary, all vector representations worked well and got similar f1 scores. The feature *LexNet (original)* was again the worst, yet the score was ten points above the baseline.

Figure 4.3.4.: F1 score for the binary scenario using XGBoost. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the f1 score on the y-axis. The black bar displays the shows derivation.

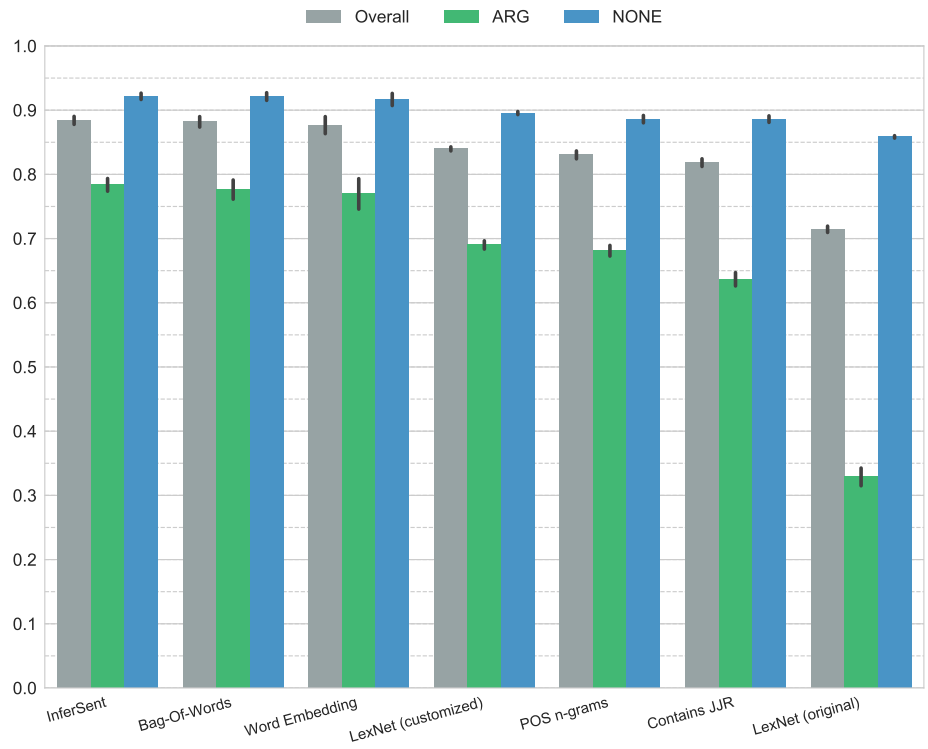


Figure 4.3.5.: Precision for the binary scenario using XGBoost. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the precision score on the y-axis. The black bar displays the standard derivation.

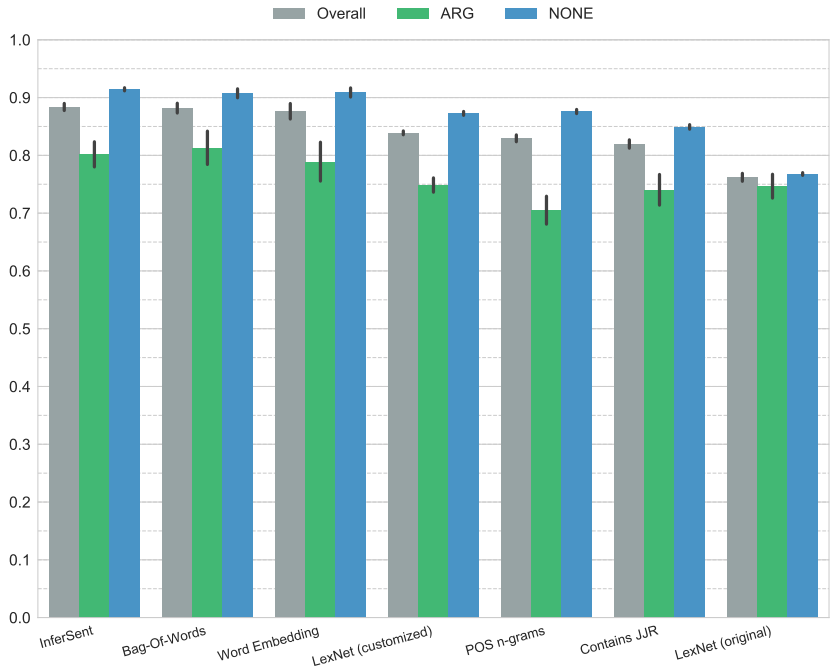
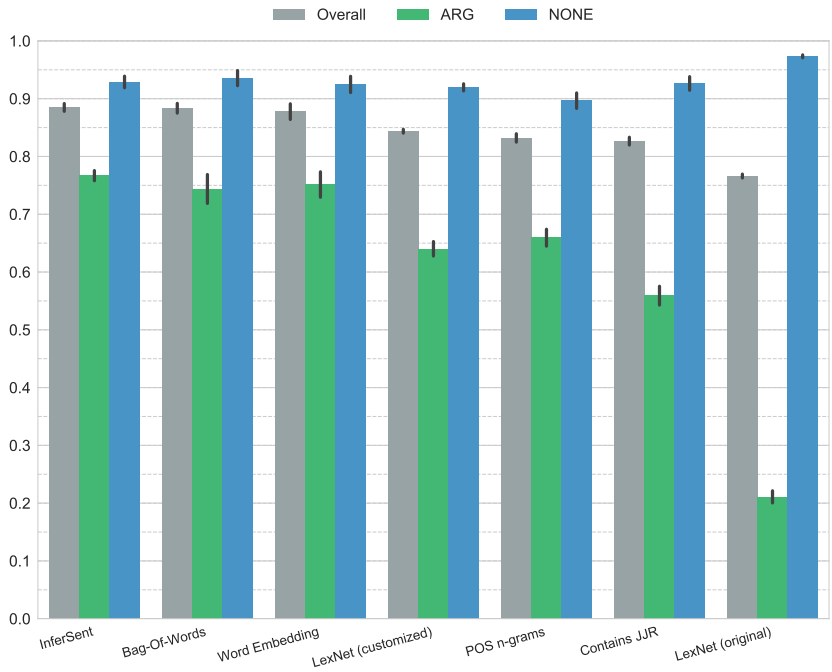


Figure 4.3.6.: Recall for the binary scenario using XGBoost. The grey bar shows the weighted. The feature names (see Section 4.2) are presented on the x-axis, the binary score on the y-axis. The black bar displays the standard derivation.



4.3.3. Error analysis

Figure 4.3.7 displays the confusion matrix for the best lexical feature in the three-class scenario (InferSent, also the best feature overall), while figure 4.3.8 shows the confusion matrix for the best syntax feature (customised LexNet). The confusion matrices of each fold per feature were summed up to create the figures.

Figure 4.3.7.: Confusion matrix for the InferSent feature using XGBoost in the three-class scenario.

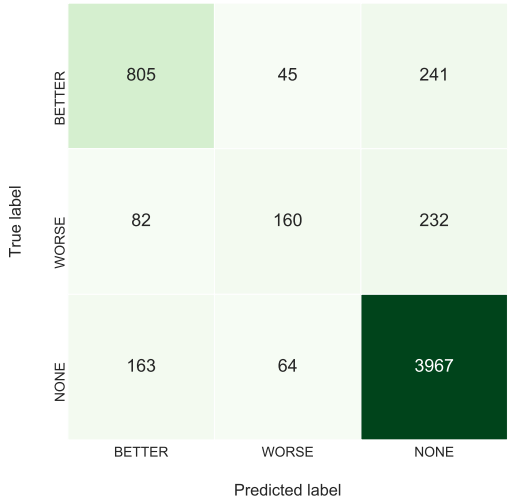
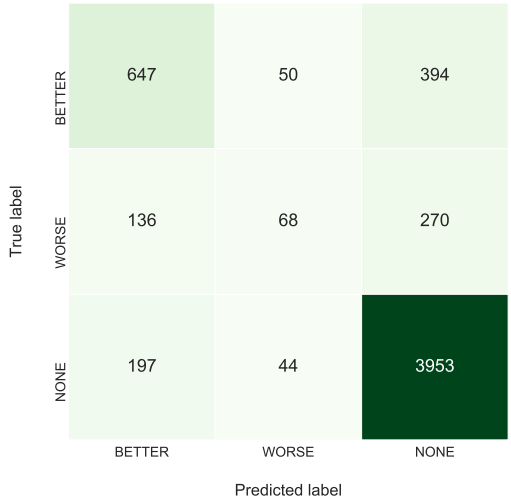


Figure 4.3.8.: Confusion matrix for the customised LexNet feature using XGBoost in the three-class scenario.



As presented above, WORSE was the hardest class to recognise. The matrices show that WORSE was more often confused with NONE than with BETTER. This is contrary to the expectations. The classes BETTER and WORSE should represent argumentative sentences. It was expected that the distinction between argumentative and not-argumentative is clearer. In total, 1311 sentences were incorrectly classified in the three-class scenario. Both features made the same errors on 607 sentences. The InferSent feature made 220 additional errors, while the LexNet feature made 484. Surprisingly, the majority of errors was made on sentences with a high confidence. Four-hundred-twenty-five of the shared errors were made on sentences with a confidence of one. InferSent made 156 errors on highly confident sentences, while LexNet made 356. Examples on errors made solely by the InferSent feature are given in Table 4.3.5. The first two sentences look comparative, but they are questions. As stated in the guidelines, all questions should be labelled as NONE, but InferSent frequently classified questions as comparative. Sentences three and four are comparative, but have no clear winner of the comparison. However, only sentences with clear winners should be labelled as BETTER or WORSE. InferSent was not able to learn this restriction. Sentence six contains three negative words. Sentence seven is hard to classify, as it does not contain any cue word.

Table 4.3.6 shows examples for errors exclusively made by the LexNet feature. As

Table 4.3.5.: Example sentences for errors made by XGBoost in the three-class scenario with the InferSent feature. The objects of interest are printed **bold**. Confidence shows the confidence of the annotators and is calculated as *judgments for majority class / total judgments*.

	Sentence	Predicted	Gold	Confidence
1	Is **Python** better than **Perl**?	BETTER	NONE	0.6
2	Is **Microsoft** better because of **Apple**?	BETTER	NONE	1.0
3	**Microsoft** is the devil but **Sony** truly isn't any better.	WORSE	NONE	1.0
4	**Python** is much better suited as a "glue" language, while **Java** is better characterized as a low-level implementation language.	BETTER	NONE	1.0
5	Its Azure PaaS/IaaS platform hasn't overtaken **Amazon** yet in market share, but **Microsoft** has enjoyed nine straight quarters of growth at 10 percent or better	NONE	WORSE	1.0
6	arrrggghh...**Python** is a terrible language - only Perl sucks worse.	WORSE	BETTER	1.0
7	Good to see again a **Renault** ahead of a **Ferrari**.	NONE	BETTER	1.0

described in Section 4.2, 399 of the sentences did not get a dependency path. However, only 36 of the wrongly classified sentences did not have a path.

It is salient that the LexNet feature made errors on fairly simple sentences like the first one in Table 4.3.6. While InferSent's errors can be coarsely grouped, the errors made by LexNet seem more random. It is assumed that the amount of training data for the neural network encoder is not big enough to create expressive embeddings. However, the overall result of LexNet indicates that a encoder trained on more data will likely yield good results.

Sentences that were wrongly classified by both features are presented in Table 4.3.7. Both features predicted the same class for 477 of the 607 shared errors.

The shared errors are similar to the errors made exclusively by InferSent. Again, they can be coarsely grouped, for instance into questions (sentence one in Table 4.3.7) or sentences without a clear winner (sentence four).

In the binary scenario, 1183 errors were made. Both features made the same errors on 380 sentences. LexNet made 520 unique errors, while InferSent made 283. However, 739 errors were already made in the three-class scenario, only 444 errors were new. All in all, the analysis of the errors made in the binary scenario did not gave any new insights. Again, the majority of errors was made on sentences with a high confidence. The errors made exclusively by LexNet seem random again, while the shared errors and the errors made by InferSent have similar sources (e.g. questions, no clear winner, complex sentences).

Table 4.3.6.: Example errors made by the classifier in the three-class scenario with the LexNet feature. The objects of interest are printed **bold**. Confidence shows the confidence of the annotators and is calculated as *judgments for majority class / total judgments*.

	Sentence	Predicted	Gold	Confidence
1	Right now **Apple** is worse then **Microsoft** ever was.	BETTER	WORSE	0.8
2	**california** is a much harder place to practice law than is South **carolina**.	NONE	WORSE	1.0
3	**google+** starts out far faster than **facebook**, Twitter	NONE	BETTER	1.0
4	In FPGAs, **Integer** or fixed-point math can often run 10 to 100 times faster than **Floating-point** computations.	NONE	BETTER	1.0
5	Your **iphone** is a much better display than your desktop/**laptop**	NONE	BETTER	0.8

Table 4.3.7.: Example errors made by InferSent and LexNet in the three-label scenario. The objects of interest are printed **bold**. *Pred. IF* shows the predicted class of the InferSent feature, *Pred. LexNet* the prediction of the LexNet feature. *Conf.* shows the confidence of the annotators and is calculated as *judgments for majority class / total judgments*.

	Sentence	Pred. IF	Pred. LexNet	Gold	Conf.
1	Is a **BMW** 3 series $15,000 better than a **Ford** Focus?	BETTER	BETTER	NONE	1.0
2	**Google** is the main player now, **Microsoft** are just plain inferior in Mobile	NONE	NONE	BETTER	1.0
3	Yeah, Nvidia's **OpenCL** is not good and **CUDA** is way better.	NONE	BETTER	WORSE	1.0
4	**Python** grew out of the need for a "better" **Perl**.	WORSE	NONE	BETTER	1.0
5	**Groovy** code often looks and feels like **Java** code, but is almost always simpler and easier to use.	NONE	NONE	BETTER	0.4

4.3.4. Discussion

Section 4.3.2 only shows the results for the best performing configuration of each feature. This is, for all cases, the middle part of the sentence. Sentences which are not formed after this pattern caused wrong predictions, as presented in Section 4.3.3.

Using the full sentence worked second best. Adding the first and/or last part of the sentence did not increase the f1 score at all, no matter if the same or another representation type than the one for the middle part was used. The first and second part alone never got an f1 score above the baseline.

Replacing or removing the objects did not increase the score significantly. In most cases, the difference in the f1 score between no replacement/removal and the best replacement/removal strategy was only reflected in the third or fourth decimal place. Hence, the concrete objects are not important at all for the classification. This is also supported by the fact that adding the word vectors of the objects as features did not increase the result for any feature.

An interesting observation is that the simple bag-of-words model performs equal or better than the more complex models. The simple feature *Contains JJR* (which is only one boolean value) is able to distinguish argumentative and non-argumentative sentences in a pleasing way, yielding an f1 score fourteen points above the baseline.

The f1 scores for LexNet path embeddings show that this is a reasonable way to encode sentences. The original setup found only paths for 26% of the sentences, yet it yielded an f1 score eight points above the baseline. The customisation made it even more powerful.

No combination of vector representations increased the score in any way. It was expected that a combination of LexNet features and one of the other features like InferSent will increase the f1 score, as they encode different information (lexical and syntactial). However, this was not the case. Appending the LexNet vectors to the InferSent vectors reduced the scores.

As expected, the smallest class in the data set caused the biggest problems. Precision, recall and f1 score of `WORSE` have a high standard derivation for most features. Looking at the confusion matrices in Figure 4.3.7 and Figure 4.3.8, `WORSE` was confused with `NONE` for the majority of cases. Intuitively, a confusion between `WORSE` and `BETTER` was expected, since both classes should reflect argumentative sentences in general.

Another interesting observation is that the three-class scenario and the binary scenario made mistakes on the same sentences. In fact, the majority of mistakes made by the binary scenario were made by the three-classes scenario as well.

4.4. Evaluation with the held-out data

4.4.1. Results

The held-out data set contains 1441 sentences (1048 NONE, 273 BETTER, 119 WORSE). The classifier was trained on the complete data set used during development (5759 sentences).

The path embeddings for the held-out data were generated with the same neural network architecture as shown in Figure 4.2.1.

The results for the three-class scenario are presented in Figure 4.4.1 (f1 score), Figure 4.4.2 (precision) and Figure 4.4.3 (recall).

Figure 4.4.1.: F1 score for the three-class scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the f1 score on the y-axis.

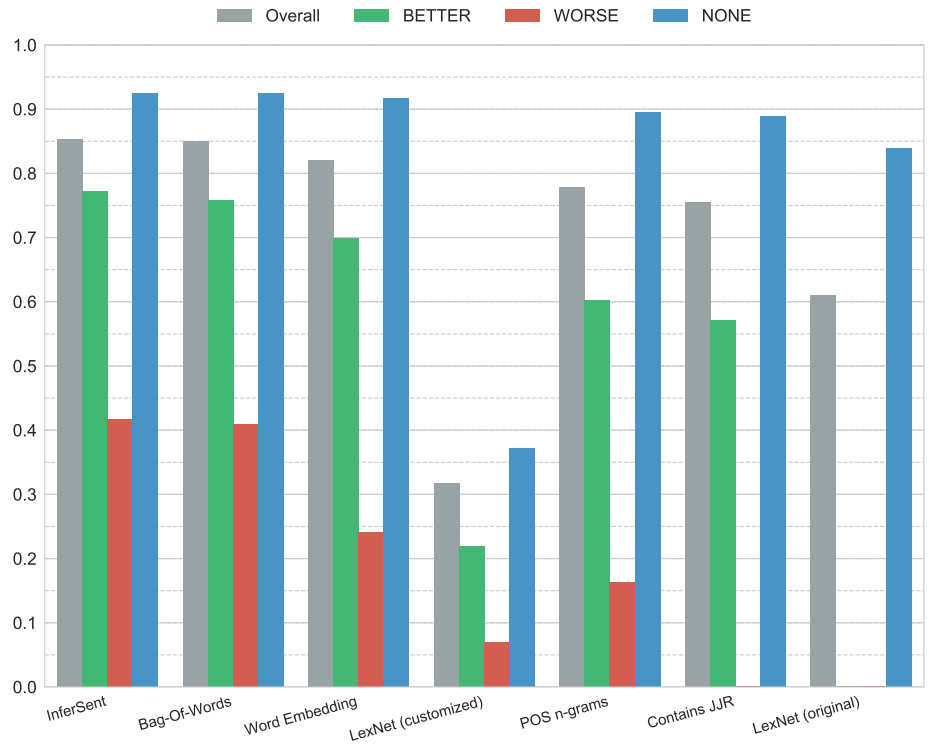


Figure 4.4.2.: Precision for the three-class scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature names (see Section 4.2) is presented on the x-axis, the precision score on the y-axis.

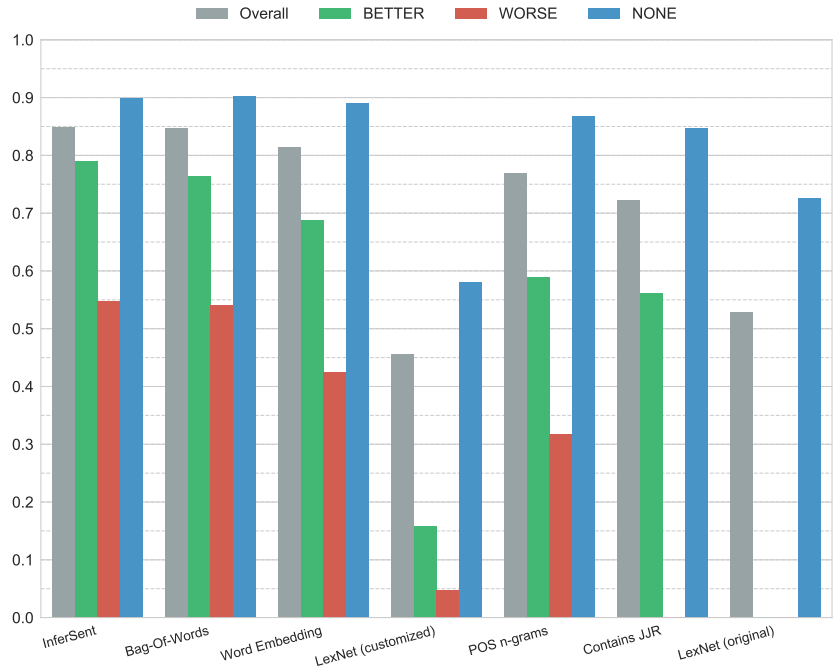
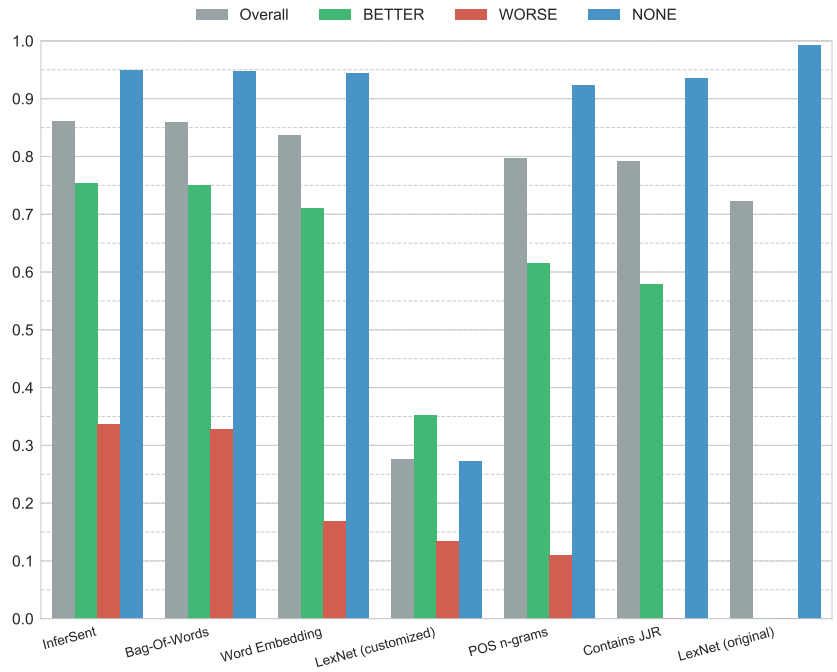


Figure 4.4.3.: Recall for the three-class scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the recall score on the y-axis.



The results for the binary scenario are presented in Figure 4.4.4 (f1 score), Figure 4.4.5 (precision) and Figure 4.4.6 (recall).

Figure 4.4.4.: F1 score for the binary scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the f1 score on the y-axis.

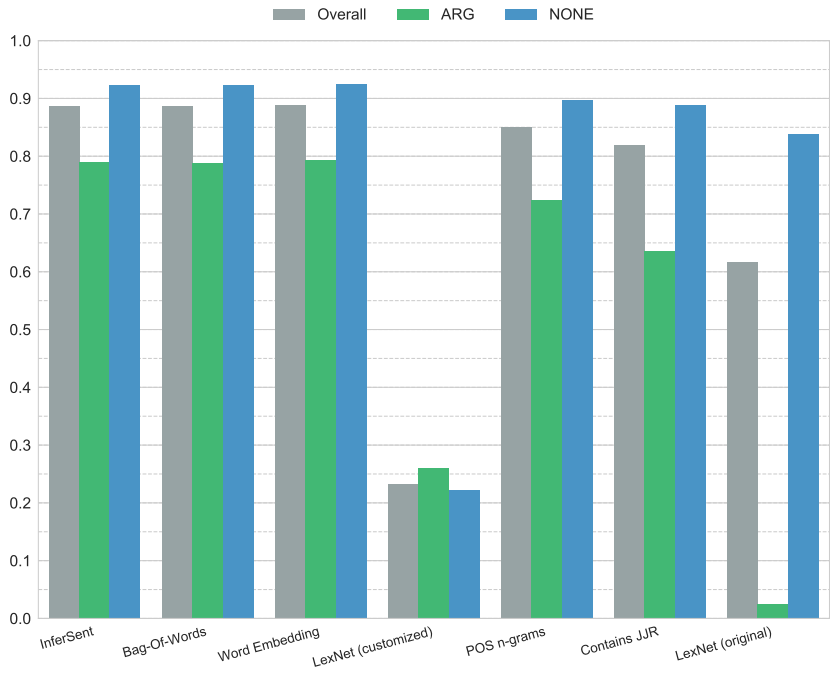


Figure 4.4.5.: Precision for the binary scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature name (see Section 4.2) are presented on the x-axis, the precision score on the y-axis.

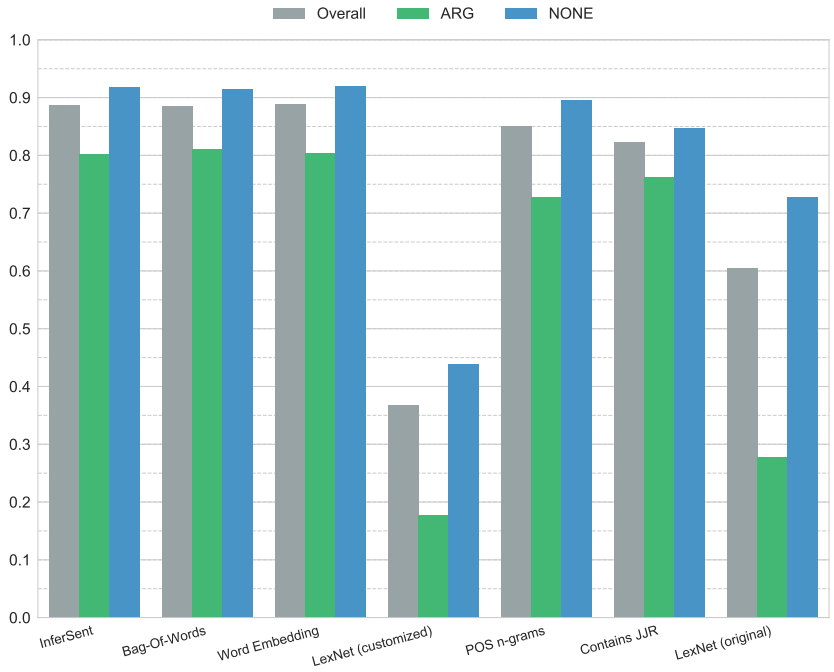
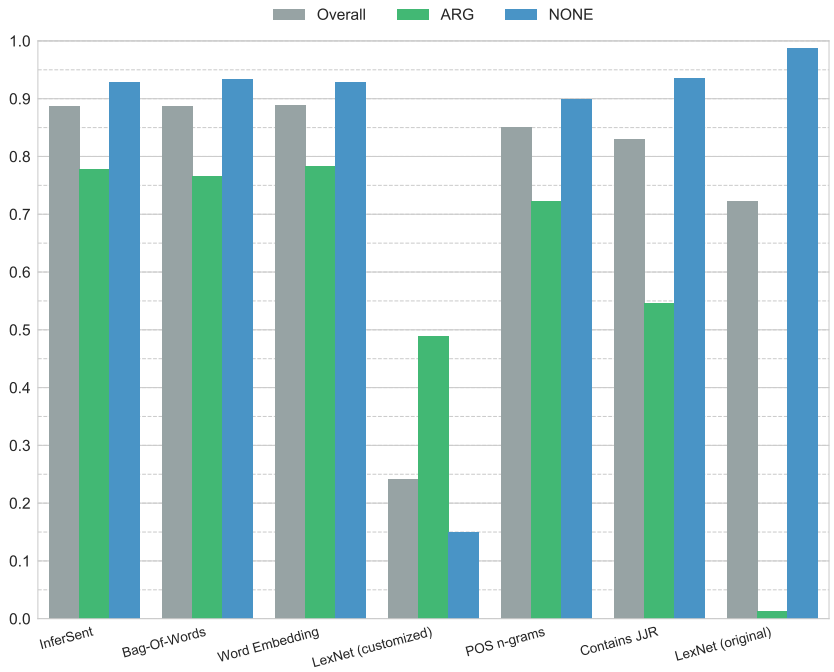


Figure 4.4.6.: Recall for the binary scenario using XGBoost with the held-out data. The grey bar shows the weighted average. The feature names (see Section 4.2) are presented on the x-axis, the recall score on the y-axis.



4.4.2. Discussion

As in Section 4.3.2, bag-of-words and InferSent got almost equal results for both scenarios. The scores of LexNet features got a significantly lower f1 score.

All features except LexNet benefited from the increased size of the training data, yet the increase of the f1 score is not larger than one or two points.

LexNet did not generalise at all. The original setup predicted `NONE` to all examples in the three-class scenario, thus it got the same f1 score as the baseline. It was able to predicts a small portion to `ARG` in the binary scenario, so that the f1 score was one point above the baseline. The customised version predicted all classes, yet the results were 27 points below the baseline for the three-class scenario and 38 points below in the binary scenario.

The problem was the size of the data set. LexNet created 2344 unique paths for the 5759 examples in the training data and 593 unique paths for the 1441 examples in the held-out data. As a result, one path represented only a few sentences. Training and held-out had only 81 paths in common. The same is true for the customised setup, were even more paths were created (4339 for the training data and 1263 for the held-out data). Training and held-out only shared 228 paths. So, from LexNets point of view, training and held-out data are highly dissimilar.

This problem can be reduced with a larger data set. The Wikipedia corpus used in [Shwartz et al., 2016] is magnitudes larger: it contains 4,682,000 articles.[7] The ratio between paths and sentences are much smaller in a corpus of this size.

The results with the training data alone (Section 4.3.2) can be explained by the fact that the paths of all training examples were learned simultaneously. In contrast to this, the held-out embeddings were learned without information from the training set. The performance of the LexNet feature increases if the path embeddings are learned on the whole data (training and held-out; see Figure 4.4.7 and Figure 4.4.8).

[7] Wikipedia Dump from 2015, `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia` (checked 15.5.2018)

Figure 4.4.7.: F1 score for the LexNet feature in the three-class scenario. This time, path embeddings were learned on all sentences (training and held-out). XGBoost was trained on the training data and tested on the held-out data.

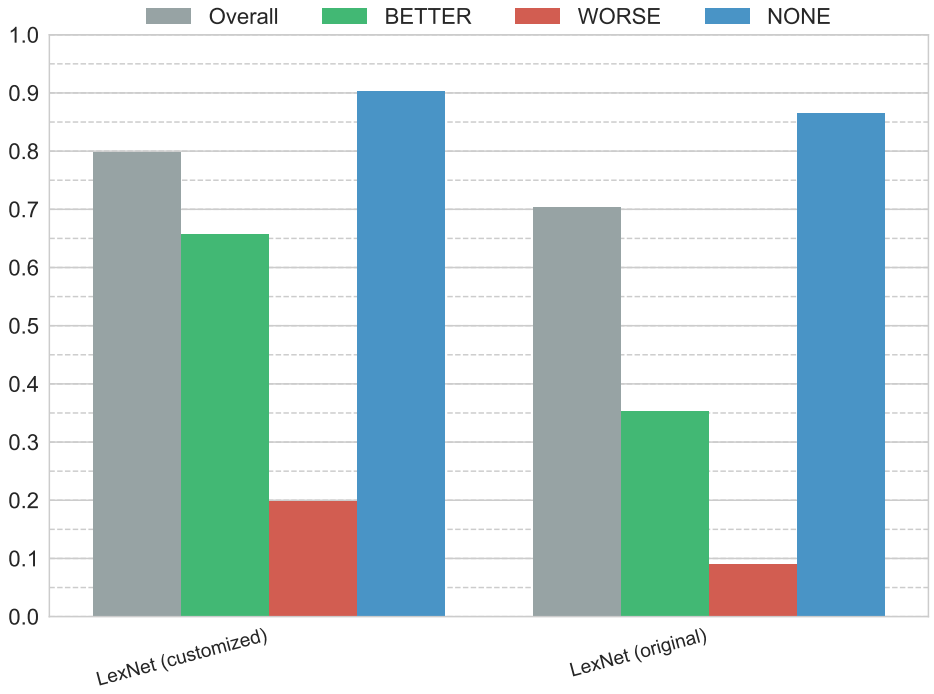
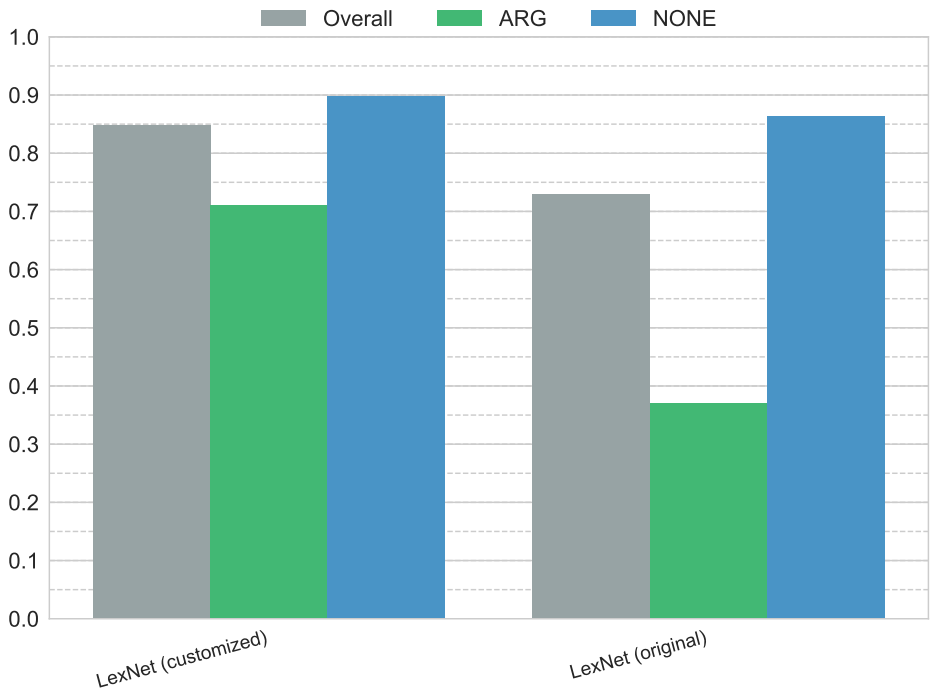


Figure 4.4.8.: F1 score for the LexNet feature in the binary scenario. This time, path embeddings were learned on all sentences (training and held-out). XGBoost was trained on the training data and tested on the held-out data.



5. Conclusion and Future Work

This thesis dealt with the problem of comparative argument mining. The first part discussed the creation of a labelled data set which contains a wide range of comparative sentences.

The second part discussed how to create a machine learning system which is able to classify the sentences in the created data set. *Gradient boosted decision trees* turned out to be the best classifier for this task. Various simple (like bag-of-words) and complex features (like sentence embeddings) achieved f1 scores at least ten points over the baseline. As presented in Section 4.3.2, the f1 score was greatly increased by some preprocessing steps. It turned out that the words between the two compared objects are most important. Features calculated with only these words outperformed all features calculated with the whole sentence. The concrete compared objects were not important at all. The removal of the objects from the sentences did not alter the results.

The simplification from a three-class problem to a binary problem (by merging the comparative classes `BETTER` and `WORSE` into one class `ARG`) increased the performance.

The final evaluation on unseen data showed that most features generalise well. All in all, the classification works satisfactory. For the three-class scenario, *InferSent* yielded an average f1 score of 0.85 (0.77 for `BETTER`, 0.42 for `WORSE` and 0.92 for `NONE`), closely followed by *bag-of-words*. *InferSent*, *bag-of-words* and *mean word embeddings* were the best features in the binary scenario. All yielded the same f1 scores (0.89 on average, 0.79 for `ARG` and 0.92 for `NONE`).

Some aspects were not covered in this thesis. Section 4.3.3 described the problem that the classifier was not able to learn the special case that questions should always belong to the class `NONE`. Future work could either remove this restriction or include a feature to identify questions. As described in Section 3.3, the data set was created on the sentence level. Because of this, no context information is available for the classification. However, the context can hold important information. For instance, the presented system does not work with a sentence like *"This is better than Java."* because the second object is missing. The preceding sentence might contain the object which is referenced by *"This"*. This would require coreference resolution[1].

Section 4.3.2 showed that the features based on LexNet yield acceptable results. It is expected that the results would increase if more data is available to create the path

[1] See Chapter 21, page 708ff. of [Martin and Jurafsky, 2009]

embeddings. In [Shwartz et al., 2016] and [Shwartz and Dagan, 2016], the systems were trained on a Wikipedia corpus, which is magnitudes larger than the 7199 sentences from the corpus created in this thesis. One (costly) approach for future work is to annotate more data. Another approach could sample new sentences from the index, by using patterns like *"is better than"* or *"is worse than"*. The quality would not be as good as with manually labelled data, but this might be compensated by the neural network if it is trained long enough.

The results in Section 4.4.1 show that several features generalise well. The f1 score for unseen data is comparable to the scores during the development phase. Yet, the system was not tested in a real world application. For example, a comparison search engine that takes two objects as the input and returns all comparisons. In a next step, the search engine could inspect the retrieved companions and extract compared properties and the like.

A. Detailed Classification Results

A.1. Feature Experiments

The following shows the classification result for each feature. Each feature was tested with five stratified folds. The result is presented as the average out of five folds with standard derivation. The class ARG is the union of BETTER and WORSE.

Table A.1.1.: Bag-of-words feature (three-class scenario). The presence of all unigrams in the corpus are represented as binary features.

	precision	recall	f1 score
BETTER	0.79 ±0.02	0.70 ±0.03	0.74 ±0.01
WORSE	0.62 ±0.06	0.36 ±0.05	0.46 ±0.05
NONE	0.89 ±0.01	0.95 ±0.01	0.92 ±0.00
average	0.85 ±0.01	0.86 ±0.00	0.85 ±0.01

Table A.1.2.: Bag-of-words feature (binary scenario). The presence of all unigrams in the corpus are represented as binary features.

	precision	recall	f1 score
ARG	0.78 ±0.03	0.79 ±0.03	0.78 ±0.0)
NONE	0.92 ±0.01	0.92 ±0.01	0.92 ±0.01
average	0.88 ±0.01	0.88 ±0.01	0.88 ±0.01

Table A.1.3.: InferSent (sentence embeddings) feature (three-class scenario).

	precision	recall	f1 score
BETTER	0.78 ±0.03	0.71 ±0.03	0.74 ±0.02
WORSE	0.60 ±0.03	0.28 ±0.05	0.39 ±0.04
NONE	0.89 ±0.00	0.96 ±0.01	0.92 ±0.00
average	0.84 ±0.01	0.86 ±0.01	0.84 ±0.01

Table A.1.4.: InferSent (sentence embeddings) feature (binary scenario).

	precision	recall	f1 score
ARG	0.82 ±0.02	0.75 ±0.01	0.79 ±0.01
NONE	0.91 ±0.00	0.94 ±0.01	0.92 ±0.00
average	0.89 ±0.01	0.89 ±0.01	0.89 ±0.01

Table A.1.5.: Mean word embeddings (three-class scenario). All GloVe word vectors of a sentence were summed up and divided by the number of words in the sentence.

	precision	recall	f1 score
BETTER	0.70 ±0.03	0.73 ±0.02	0.72 ±0.01
WORSE	0.45 ±0.09	0.15 ±0.04	0.22 ±0.05
NONE	0.89 ±0.00	0.95 ±0.01	0.92 ±0.00
average	0.82 ±0.01	0.84 ±0.00	0.82 ±0.00

Table A.1.6.: Mean word embeddings (binary class scenario). All GloVe word vectors of a sentence were summed up and divided by the number of words in the sentence.

	precision	recall	f1 score
ARG	0.77 ±0.03	0.78 ±0.02	0.77 ±0.02
NONE	0.92 ±0.01	0.91 ±0.01	0.91 ±0.01
average	0.88 ±0.01	0.88 ±0.01	0.88 ±0.01

Table A.1.7.: Part-of-speech n-gram feature (three-class scenario). The presence of the 500 most frequent part-of-speech bi-, tri- and four-grams were represented as binary features.

	precision	recall	f1 score
BETTER	0.61 ±0.03	0.56 ±0.02	0.58 ±0.02
WORSE	0.20 ±0.05	0.09 ±0.03	0.12 ±0.04
NONE	0.86 ±0.01	0.93 ±0.01	0.89 ±0.00
average	0.76 ±0.01	0.79 ±0.01	0.77 ±0.01

Table A.1.8.: Part-of-speech n-gram feature (binary scenario). The presence of the 500 most frequent part-of-speech bi-, tri- and four-grams were represented as binary features.

	precision	recall	f1 score
ARG	0.69 ±0.02	0.68 ±0.01	0.69 ±0.01
NONE	0.88 ±0.00	0.89 ±0.01	0.88 (0.01)
average	0.83 ±0.01	0.83 ±0.01	0.83 ±0.01

Table A.1.9.: Contains JJR feature which represents the presence of a comparative adjective in the sentence (three-class scenario).

	precision	recall	f1 score
BETTER	0.56 ±0.02	0.61 ±0.02	0.58 ±0.01
WORSE	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
NONE	0.85 ±0.00	0.92 ±0.01	0.88 ±0.00
average	0.72 ±0.00	0.79 ±0.01	0.75 ±0.01

Table A.1.10.: Contains JJR feature which represents the presence of a comparative adjective in the sentence (binary scenario).

	precision	recall	f1 score
ARG	0.75 ±0.03	0.55 ±0.02	0.63 ±0.01
NONE	0.85 ±0.00	0.93 ±0.01	0.89 ±0.01
average	0.82 ±0.01	0.83 ±0.01	0.82 ±0.01

Table A.1.11.: LexNet path embeddings with a maximum length of four and restrictions of the edge direction (three-class scenario). This setup is equal to the original setup in [Shwartz and Dagan, 2016]

	precision	recall	f1 score
BETTER	0.66 ±0.04	0.21 ±0.02	0.31 ±0.03
WORSE	0.44 ±0.14	0.04 ±0.01	0.08 ±0.02
NONE	0.76 ±0.00	0.98 ±0.00	0.86 ±0.00
average	0.72 ±0.01	0.76 ±0.00	0.69 ±0.01

Table A.1.12.: LexNet path embeddings with a maximum length of four and restrictions of the edge direction (binary scenario). This setup is equal to the original setup in [Shwartz and Dagan, 2016]

	precision	recall	f1 score
ARG	0.73 ±0.02	0.21 ±0.01	0.33 ±0.01
NONE	0.77 ±0.00	0.97 ±0.00	0.86 ±0.00
average	0.76 ±0.01	0.76 ±0.00	0.71 ±0.00

Table A.1.13.: LexNet path embeddings with a maximum length of sixteen and no restrictions of the edge direction (three-class scenario).

	precision	recall	f1 score
BETTER	0.68 ±0.02	0.54 ±0.04	0.60 ±0.02
WORSE	0.34 ±0.06	0.15 ±0.01	0.21 ±0.02
NONE	0.86 ±0.01	0.96 ±0.01	0.90 ±0.00
average	0.78 ±0.00	0.81 ±0.00	0.79 ±0.00

Table A.1.14.: LexNet path embeddings with a maximum length of sixteen and no restrictions of the edge direction (binary scenario).

	precision	recall	f1 score
ARG	0.74 ±0.01	0.65 ±0.01	0.69 ±0.01
NONE	0.87 ±0.00	0.92 ±0.01	0.89 (0.00)
average	0.84 ±0.00	0.84 ±0.00	0.84 ±0.00

A.2. Final Held-Out Experiments

average wie bei classification report

Table A.2.1.: Bag-of-words feature (three-class scenario). The presence of all unigrams in the corpus are represented as binary features.

	precision	recall	f1 score
BETTER	0.76	0.75	0.76
WORSE	0.54	0.33	0.41
NONE	0.90	0.95	0.92
average	0.85	0.86	0.85

Table A.2.2.: Bag-of-words feature (binary scenario). The presence of all unigrams in the corpus are represented as binary features.

	precision	recall	f1 score
ARG	0.81	0.77	0.79
NONE	0.91	0.93	0.92
average	0.89	0.89	0.89

Table A.2.3.: InferSent (sentence embeddings) feature (three-class scenario).

	precision	recall	f1 score
BETTER	0.79	0.75	0.77
WORSE	0.55	0.34	0.42
NONE	0.90	0.95	0.92
average	0.85	0.86	0.85

Table A.2.4.: InferSent (sentence embeddings) feature (binary scenario).

	precision	recall	f1 score
ARG	0.80	0.78	0.79
NONE	0.92	0.93	0.92
average	0.89	0.89	0.89

Table A.2.5.: Mean word embeddings (three-class scenario). All GloVe word vectors of a sentence were summed up and divided by the number of words in the sentence.

	precision	recall	f1 score
BETTER	0.69	0.71	0.70
WORSE	0.43	0.17	0.24
NONE	0.89	0.94	0.92
average	0.81	0.84	0.82

Table A.2.6.: Mean word embeddings (binary-class scenario). All GloVe word vectors of a sentence were summed up and divided by the number of words in the sentence.

	precision	recall	f1 score
ARG	0.80	0.78	0.79
NONE	0.92	0.93	0.92
average	0.89	0.89	0.89

Table A.2.7.: Part-of-speech n-gram feature (three-class scenario). The presence of the 500 most frequent part-of-speech bi-, tri- and four-grams were represented as binary features.

	precision	recall	f1 score
BETTER	0.59	0.62	0.60
WORSE	0.32	0.11	0.16
NONE	0.87	0.92	0.89
average	0.77	0.80	0.78

Table A.2.8.: Part-of-speech n-gram feature (binary scenario). The presence of the 500 most frequent part-of-speech bi-, tri- and four-grams were represented as binary features.

	precision	recall	f1 score
ARG	0.73	0.72	0.72
NONE	0.90	0.90	0.90
average	0.85	0.85	0.85

Table A.2.9.: Contains JJR feature which represents the presence of a comparative adjective in the sentence (three-class scenario).

	precision	recall	f1 score
BETTER	0.56	0.58	0.57
WORSE	0.00	0.00	0.00
NONE	0.85	0.94	0.89
average	0.72	0.79	0.76

Table A.2.10.: Contains JJR feature which represents the presence of a comparative adjective in the sentence (binary scenario).

	precision	recall	f1 score
ARG	0.76	0.55	0.64
NONE	0.85	0.94	0.89
average	0.82	0.83	0.82

Table A.2.11.: LexNet path embeddings with a maximum length of four and restrictions of the edge direction (three-class scenario). This setup is equal to the original setup in [Shwartz and Dagan, 2016]

	precision	recall	f1 score
BETTER	0.00	0.00	0.00
WORSE	0.00	0.00	0.00
NONE	0.73	0.99	0.84
average	0.53	0.72	0.61

Table A.2.12.: LexNet path embeddings with a maximum length of four and restrictions of the edge direction (binary scenario). This setup is equal to the original setup in [Shwartz and Dagan, 2016]

	precision	recall	f1 score
ARG	0.28	0.01	0.02
NONE	0.73	0.99	0.84
average	0.61	0.72	0.62

Table A.2.13.: LexNet path embeddings with a maximum length of sixteen and no restrictions of the edge direction (three-class scenario).

	precision	recall	f1 score
BETTER	0.16	0.35	0.22
WORSE	0.05	0.13	0.07
NONE	0.58	0.27	0.37
average	0.46	0.28	0.32

Table A.2.14.: LexNet path embeddings with a maximum length of sixteen and no restrictions of the edge direction (binary scenario).

	precision	recall	f1 score
ARG	0.18	0.49	0.26
NONE	0.44	0.15	0.22
average	0.37	0.24	0.23

Bibliography

[Aker et al., 2017] Aker, A., Sliwa, A., Ma, Y., Lui, R., Borad, N., Ziyaei, S., and Ghobadi, M. (2017). What works and what does not: Classifier and feature analysis for argument mining. In *Proceedings of the 4th Workshop on Argument Mining*, pages 91–96, Copenhagen, Denmark. Association for Computational Linguistics.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Bowman et al., 2015] Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642. Association for Computational Linguistics.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. Association for Computing Machinery.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, ACM International Conference Proceeding Series, pages 160–167, Helsinki, Finland. Association for Computing Machinery.

[Conneau et al., 2017] Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natu-*

ral Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

[Daxenberger et al., 2017] Daxenberger, J., Eger, S., Habernal, I., Stab, C., and Gurevych, I. (2017). What is the essence of a claim? cross-domain claim identification. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2055–2066, Copenhagen, Denmark. Association for Computational Linguistics.

[Dusmanu et al., 2017] Dusmanu, M., Cabrio, E., and Villata, S. (2017). Argument mining on twitter: Arguments, facts and sources. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2317–2322, Copenhagen, Denmark. Association for Computational Linguistics.

[Eckle-Kohler et al., 2015] Eckle-Kohler, J., Kluge, R., and Gurevych, I. (2015). On the role of discourse markers for discriminating claims and premises in argumentative discourse. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2236–2242, Lisbon, Portugal. Association for Computational Linguistics.

[Fiszman et al., 2007] Fiszman, M., Demner-Fushman, D., Lang, F. M., Goetz, P., and Rindflesch, T. C. (2007). Interpreting comparative constructions in biomedical text. In *Biological, translational, and clinical language processing, BioNLP@ACL 2007, Prague, Czech Republic, June 29, 2007*, pages 137–144, Prague, Czech Republic. Association for Computational Linguistics, Association for Computational Linguistics.

[Friedman et al., 2009] Friedman, J., Hastie, T., and Tibshirani, R. (2009). *The Elements of Statistical Learning – Data Mining, Inference, and Prediction*, volume 2 of *Springer series in statistics*. Springer.

[Goodfellow et al., 2016] Goodfellow, I. J., Bengio, Y., and Courville, A. C. (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press. `http://www.deeplearningbook.org`.

[Gupta et al., 2017] Gupta, S., Mahmood, A. S. M. A., Ross, K., Wu, C. H., and Vijay-Shanker, K. (2017). Identifying comparative structures in biomedical text. In *BioNLP 2017, Vancouver, Canada, August 4, 2017*, pages 206–215, Vancouver, Canada. Association for Computational Linguistics.

[Harris, 1954] Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

[Hill et al., 2016] Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies, San Diego California, USA, June 12-17, 2016, pages 1367–1377, San Diego, California, USA. Association for Computational Linguistics.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

[Kiros et al., 2015] Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3294–3302. Neural Information Processing Systems Conference.

[Lin et al., 2017] Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

[Lippi and Torroni, 2016] Lippi, M. and Torroni, P. (2016). Argumentation mining: State of the art and emerging trends. *ACM Trans. Internet Technol.*, 16(2):10:1–10:25.

[Liu et al., 2016] Liu, Y., Sun, C., Lin, L., and Wang, X. (2016). Learning natural language inference using bidirectional lstm model and inner-attention. *arXiv preprint arXiv:1605.09090*.

[Martin and Jurafsky, 2009] Martin, J. H. and Jurafsky, D. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2nd edition.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, Lake Tahoe, Nevada, United States. Neural Information Processing Systems Conference.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, Boston.

[Panchenko et al., 2018] Panchenko, A., Ruppert, E., Faralli, S., Ponzetto, S. P., and Biemann, C. (2018). Building a web-scale dependency-parsed corpus from commoncrawl. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*, Miyazaki, Japan. European Language Resources Association.

[Park and Blake, 2012] Park, D. H. and Blake, C. (2012). Identifying comparative claim sentences in full-text scientific articles. In *Proceedings of the Workshop on Detecting Structure in Scholarly Discourse*, pages 1–9. Association for Computational Linguistics.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Shwartz and Dagan, 2016] Shwartz, V. and Dagan, I. (2016). The roles of path-based and distributional information in recognizing lexical semantic relations. *CoRR*, abs/1608.05014.

[Shwartz et al., 2016] Shwartz, V., Goldberg, Y., and Dagan, I. (2016). Improving hypernymy detection with an integrated path-based and distributional method. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, volume 1. Association for Computational Linguistics.

[Šnajder, 2017] Šnajder, J. (2017). Social media argumentation mining: The quest for deliberateness in raucousness.

[Stab and Gurevych, 2014] Stab, C. and Gurevych, I. (2014). Identifying argumentative discourse structures in persuasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 46–56, Doha, Qatar. Association for Computational Linguistics.

[Wieting et al., 2015] Wieting, J., Bansal, M., Gimpel, K., and Livescu, K. (2015). Towards universal paraphrastic sentence embeddings.

[Zhao et al., 2015] Zhao, H., Lu, Z., and Poupart, P. (2015). Self-adaptive hierarchical sentence model. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 4069–4076, Buenos Aires, Argentina. AAAI Press.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemaß aus Veroffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prufungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den