Sentiment analysis of a German Twitter-Corpus

Malte Flender, Carsten Gips

FH Bielefeld - Campus Minden,
Artilleriestraße 9, 32427 Minden, Germany
{malte.flender,carsten.gips}@fh-bielefeld.de
http://www.fh-bielefeld.de/

Abstract. The amount of easily accessible texts in different languages on the internet grows daily and so the effort and need to organize these texts grows as well. An automated process is needed to extract useful information, like a sentiment, from this amount of published text. This paper deals with the extraction of a sentiment, divided into three classes, from tweets written in German language. Different machine learning algorithms and a variety of preprocessing steps are compared to find the optimal combination. While most work in this field aims at English-language tweets, this paper adapts and transfers these ideas to German-language tweets. The results are compared to the findings of other projects designed for English-language tweets. Further it is shown that the impact of the chosen feature encoding on the results is most significant.

Keywords: Twitter, Sentiment Analysis, Machine Learning, Classification, German

1 Introduction

The popularity of online-social-media, especially microblogging services like Twitter, has vastly increased in the last few years. The idea behind microblogging services is, that a person can write a short statement and post it online. These statements within Twitter are called tweets. A tweet can be shared with and seen by the writer's friends and community.

These tweets often contain a sentiment regarding a certain topic. A single tweet on its own is not very expressive. But if a (topic related) web-crawler collects a significant amount of these tweets, a sentiment analysis can provide useful information. For example, the sentiment of potential customers regarding a certain product, like a smart phone, or the sentiment towards a whole company. Depending on the topic, this information can be used in a wide variety of applications.

In contrast to other forms of text, e.g. movie reviews [7], a tweet is extremely short, at most 140 characters; the language used is so informal that it often contains spelling errors; the posts deal with a lot of different topics, and the point of view is rather subjective. These circumstances make it harder to determine an accurate classification of this kind of text compared to well-written texts, like movie reviews.

Most of the work done in this field deals with tweets written in English, e.g. [2,3]. Since Twitter is also popular in non-English speaking countries, it can be used as a source for informal texts in German language. A use case for such a classification could be a news board displaying the sentiment of the students towards the university or certain classes.

Using the German language in sentiment analysis is considered especially challenging, as the language rules are more complex compared to the English-language, e.g. there is more variety (e.g. »the« vs. »der, die, das«), and nouns must be capitalized. Furthermore sentence syntax is considered more complex in general.

This paper deals with the question whether and how methods of sentiment analysis on English-language tweets can be transferred to German-language tweets. We present a brief overview of related work on English-language texts. Different pre-processing and feature extraction steps are investigated. Four classification algorithms are tested in a complex series of experiments on German tweets. The results are compared to the previous work done on English tweets and it is shown that the feature extraction is one of the most important parts of the whole process in order to gain a high classification quality.

2 Related Work

Many results have been published regarding sentiment analysis on English tweets and movie reviews. This section deals with the results of these papers in more detail.

The first naive attempts of sentiment classification lead one to select two list of words, one corresponding to positive sentiments and one to negative sentiments. This method, as described by Pang et al. in [7] works poorly on movie reviews, at an accuracy of 58 % to 64 %, with the random-choice baseline performing at a 50 % accuracy, since only two equally distributed classes were used. In a second attempt Pang et al. used machine learning methods to improve their results. They used the Support Vector Machine (SVM), Naive Bayes (NB) and Maximum Entropy (ME) algorithms. Furthermore, they used a three-fold cross validation on 700 positive and 700 negative reviews. Their feature vector consists of the frequency or the presence of the most common 16165 uni- and bi-grams. In this case the presence feature vector led to better results. A variety of different methods was tested: uni-grams + bi-grams, bi-grams, uni-grams, part of speech (POS) tagging and more. The best overall performance was achieved with the SVM classifier and the uni-grams presence as feature vector. This approach turned out to be accurate in 82.9 % of the given cases, which was significantly better than the naive approach.

A completely different approach was described by Pang et al. in [6]. The idea here was to check every sentence in the movie review with a machine learning algorithm, like NB or a SVM, to see if the movie review contained a sentiment or was objective. The objective sentences were discarded. The result was a much sparser data set, with almost no information loss. This data set, which only consisted of two classes, was used as the input of the second stage, in which a graph was built based on the given data. Within this graph a minimum cut was found, which was used to determine the sentiment. Compared to using a simple NB classifier, the accuracy increased from 82.8 % up to 86.4 % for NB used in both steps. The minimum cut approach outperformed the NB and the SVM with 86.4 % (cut) vs. 85.2 % (NB) and 86.15 % (cut) vs. 85.45 % (SVM).

The two above mentioned discussions focused on movie reviews, however Go et al. dealt with Twitter posts as data supply in [2]. They used two sentiments, »positive« and »negative« to classify the tweets, which were written in English. They used the NB, ME and SVM as classifiers and emoticons, like :-) or :-(of the tweets as noisy labels. Their feature vector consists of uni-grams, bi-grams, uni-grams + bi-grams and uni-grams with POS tags. To reduce the corpus size they replaced user names, URL links and repeated letters, like »huuungry«. This reduction led to a size decrease of 45.85 %. The used baseline consists of a word count, in which the words originated from lists for both sentiments. For the uni-gram case the baseline was at 65.2 %; the classifiers result was in NB 81.3 %, ME 80.5 % and SVM 82,2 %. The overall best performance was achieved with uni- and bi-grams and ME with 83.0 % accuracy.

3 Corpus

Since the corpus is extremely important to the performance of the classification task, it is necessary to have a closer look at the corpus which is used. Twitter corpora written in English can be found quite often, but a fully annotated German corpus is harder to find.

There are some German corpora, like the one provided by Bütow et al. in [1], but this corpus only contains news statements, which differ from Twitter statements in various ways. A news text is written more carefully and uses a completely different ductus; furthermore it rarely contains any emoticons, slang words, spelling errors or repeated letters at all.

The corpus provided by Narr et al. [5] contains labeled tweets in multiple languages. For this task we use the German-language part of the corpus, i.e. 1800 German-language tweets labeled as either »positive«, »negative« or »neutral«. This is a difference to the above mentioned discussions [2,6,7], where only »positive« and »negative« classes were utilized. These labels were assigned to the tweets by three different humans via Amazon Mechanical Turk, which means that in some cases the same tweet was labeled differently. To accommodate to this fact, the corpus is split into three data sets. The first set contains all tweets with different labels, now referred to as »agree1«. The second set contains only tweets in which at least two of the labels matched (»agree2«). The last set consists only of tweets in which all three labels match (»agree3«). The »agree2« data set still contains 1719 tweets, while in the »agree3« set only 958 tweets are found. So »agree3« $\subset$ »agree2« $\subset$ »agree1«. This results in an inter-human-agreement of 1719/1800 = 95.50 % and 958/1800 = 53.22 % respectively. In this corpus some preprocessing has already been done. Links within the tweets have been replaced with a »~http« tag, also user names have been replaced by »@user«.

This corpus was used in [5] as test-data. Training-data was gained by using another tweet-corpus, labeled using the emoticons as noisy labels. Then a Naive Bayes classifier was used to classify the tweets in this corpus and achieved an accuracy of up to 79.8 % on the German data set.

4 Approach

In this paper we use Python with NLTK and scikit-learn. The resulting script can be obtained under the MIT License from `https://bitbucket.org/snippets/mflender/eBKAy`.

4.1 Preprocessing

The preprocessing of the tweets contains several steps that are common to natural language processing. These steps will be described in more detail.

The corpus is UTF-8 encoded, since the German alphabet contains some non-ASCII symbols, like ö, ä, ü, ß. However, there are few entries containing non-UTF-8 characters. These lines are filtered first. The next step is to normalize the tweets, i.e. to change the tweets to lower case. This step can be skipped to evaluate the influence of the normalization. To split the tweets into its words and punctuation marks two different tokenizers are used: The WordPunctTokenizer, which treats a simple emoticon as a single word, and the WordTokenizer, which splits an emoticon in its components.

The next step in the process is to remove stopwords. We use a list of German stopwords provided by the NLTK framework. In addition to that list some other characters, like »&«, »)«, »(«, »=« are added to the list of stopwords. To evaluate the influence of stopword removal, this step also can be skipped. As a last preprocessing step the tweets are processed by the Snowball stemmer.

Now the tokens are converted into *n*-grams. We use several combinations of uni-, bi-, and tri-grams to include word negations and adjectives like *nicht gut* (not good) or *sogar besser als* (even better than). An overview of the preprocessing steps and its variations is given in Fig. 1.

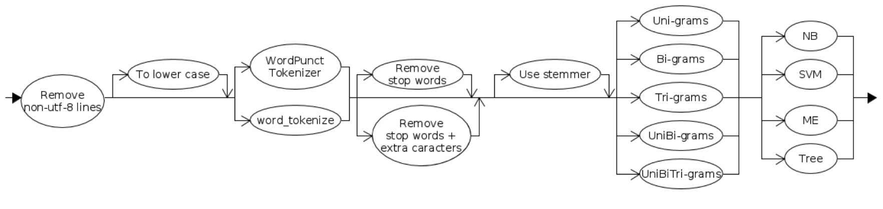

Fig. 1. Diagram of the different preprocessing and classification steps

4.2 Feature Extraction

After the pre-processing features are extracted to form the feature vector. To do so two different approaches are used. We refer to the first one as »sparse« and to the second one as »dense«. The sparse approach is discussed first:

A list of the k most common n-grams in the training set is generated and used as feature vector. All tweets in the data set are represented by an instance of the feature vector, indicating the individual occurrences of the features (»binary encoding«). This approach creates a uniform feature space but compared to the dense approach it consumes more memory.

The dense approach features the exact opposite behavior; it uses only a little memory, but creates an asymmetric feature space. For every tweet the feature vector consists of all n-grams contained in the individual tweet.

A naming scheme to uniquely describe the performed experiments is given in Table 1. This scheme is used to present the results in Table 2, Table 3 and Table 4.

SNS	→ Use cases as they were
LOW	→ Set everything to lower case
WPT	→ Use the WordPunctTokenizer
WT	→ Use the WordTokenizer
STP	→ With stopwords and extended stopwords filtered
nEXSTP	→ Only with stopwords filtered
nSTP	→ Without any stopwords filtered
STM	→ With stemmer
nSTM	→ Without stemmer
Uni	→ Use uni-grams
Bi	→ Use bi-grams
Tri	→ Use tri-grams
UniBi	→ Use uni- and bi-grams
UniBiTri	→ Use uni-, bi- and tri-grams
DNS	→ Use the dense feature vector
SPR(k)	→ Use the sparse feature vector with the k most common n-grams

Table 1. Key to the settings symbols from Table 2, Table 3 and Table 4

4.3 Classification

The process of classification, the classifiers and their settings are now described in more detail. For all classifiers the basic approach is the same. So the general approach is discussed first.

To utilize as much data as possible a ten-fold cross validation is used. The whole corpus is shuffled before classification to prevent influencing the final results by the order of the tweets. The size of the training and test sets, the accuracy of the classification, the time that is used for the training and classification process, the F-measure and its components (precision and recall) are measured for every class. All this information is gained for every fold. The arithmetic mean of all folds is used as the final result.

For the classification four different classifiers are used:

- `sklearn.naive_bayes.MultinomialNB`,
- `sklearn.svm.LinearSVC`,
- `nltk.DecisionTreeClassifier` and
- `nltk.MaxentClassifier`.

The Multinomial NB and the NLTK decision tree classifier are used without further options. Since NLTK does not provide its own SVM implementation, the Sklearn SVM with linear kernel is used. A pretest of the different kernel settings had shown that the linear kernel performs the best on the given data set. The radial basis function kernel lacks about one percent of accuracy and the polynomial kernel performs even worse.

We use the NLTK implementation of the ME with the MEGAM[1] algorithm. It provides the same accuracy and a higher speed compared to other available ME-algorithms. The number of iterations is set to 20 in a series of pre-experiments. This setting provides the same acceptance rate as the default settings (100 iterations), but takes only a fifth of the time.

Instead of a random-choice baseline, a ZeroR classifier, which decides always for the most common class is used. Thus the percentage occurrences of this class is used as baseline; in the corpus this is the »neutral« one. The used data set is quite small, so it could be possible that in one of the folds there will be no member of the least common class (»negative«). To check upon this possibility we evaluated the proportion of every class in percent for every fold. The result of this test shows that in every fold every class is represented.

5 Evaluation & Results

For the final evaluation of the different classifiers and preprocessing steps only the »agree3« data set is used. The »agree1« and »agree2« data sets are not used to avoid the problem of which sentiment to use in case all the Mechanical Turk workers disagreed. In addition to that, when two or three humans have different opinions on the sentiment of a tweet, that tweet will most likely either have multiple sentiments or its sentiment is very ambiguous.

The »agree3« results are presented in Table 2, Table 3 and Table 4, the explanation for the used acronyms can be found in Table 1. The baseline performance for the experiments is 74.92 %. This means that 74.92 % of all tweets in the set were labeled as »neutral«, while 15.07 % where labeled as »positive« and

[1] `http://www.umiacs.umd.edu/~hal/megam/`

10.01 % as »negative«. This result for the baseline differs slightly from the one computed at [1], due to the filtering of the non-UTF8 lines.

Even without considering variations of parameter k, there are about 120 different combinations in the proposed setup, that one can experiment with. These are too many combinations to try out. Therefore, we define a basic setup for both proposed feature vector variants and examine systematically from this scenarios the combination and variation of different parameters and steps to find an optimal setting. Experiment 4 will serve as a standard for the sparse vector and experiment 21 for the dense feature vector. Both standards use all possible preprocessing steps and where tested first. All other variations are employed step by step originating from experiment 4 and 21. For the sparse standard we use $k = 1000$ since it provides a good balance between train + test time and classification accuracy. Kouloumpis et al. also reported this value as useful for sentiment analysis on English-language tweets in [4].

	Used settings (see Table 1)						NB	SVM	ME	Tree
(1)	SPR(1)	STP	STM	WPT	UniBi	LOW	74.92	74.92	74.92	74.92
(2)	SPR(10)	STP	STM	WPT	UniBi	LOW	74.92	74.92	**75.02**	74.92
(3)	SPR(100)	STP	STM	WPT	UniBi	LOW	74.92	75.02	74.81	**75.13**
(4)	SPR(1000)	STP	STM	WPT	UniBi	LOW	75.66	**78.08**	71.97	77.76
(5)	SPR(2000)	STP	STM	WPT	UniBi	LOW	70.28	77.34	75.86	**77.55**
(6)	SPR(3000)	STP	STM	WPT	UniBi	LOW	64.17	**78.61**	75.45	77.13
(7)	SPR(4000)	STP	STM	WPT	UniBi	LOW	60.59	**79.34**	72.81	76.82
(8)	SPR(5000)	STP	STM	WPT	UniBi	LOW	60.49	**79.87**	73.55	77.97
(9)	SPR(10000)	STP	STM	WPT	UniBi	LOW	49.11	**83.45**	75.44	82.82
(10)	SPR(10307)	STP	STM	WPT	UniBi	LOW	47.53	**83.13**	76.92	82.51

Table 2. The results of the k variations for the sparse classification from the »agree3« data set compiled in a table: Accuracy in percent, where the best results per line is written in bold letters

Experiments 1 to 10 of Table 2 show the accuracy affected by the number of features. Experiment 1 only performs with baseline accuracy (74.92 %), since there is not enough information to gain from only one feature. Surprisingly, with just 10 features, the ME classifier is able to perform a little better than baseline. The NB needs more than 100 features to make a decision and its accuracy decreases when more features are available. The corpus contains 10307 distinct features, which is the upper bound for this series of tests.

Experiments 11 to 14 in comparison to experiment 4 show the influence of the different n-grams on the accuracy. Most classifiers seem to perform best with the combination of uni- and bi-grams. The exception here is the ME with uni-grams, which performs around 3 % better.

Experiments 4, 15 and 16 show the effect of the stopword removal on the classification. In most cases the accuracy decreases when no stopwords are used. The notable exception here is the ME classifier where the removal of stopwords

	Used settings (see Table 1)						NB	SVM	ME	Tree
(11)	SPR(1000)	STP	STM	WPT	Uni	LOW	72.29	75.02	**75.13**	75.02
(12)	SPR(1000)	STP	STM	WPT	Bi	LOW	73.13	**75.34**	63.85	75.13
(13)	SPR(1000)	STP	STM	WPT	Tri	LOW	71.65	75.13	65.76	75.13
(14)	SPR(1000)	STP	STM	WPT	UniBiTri	LOW	74.92	**76.81**	69.33	76.50
(15)	SPR(1000)	nEXSTP	STM	WPT	UniBi	LOW	75.44	76.08	70.07	**76.92**
(16)	SPR(1000)	nSTP	STM	WPT	UniBi	LOW	73.55	75.76	**76.50**	75.76
(17)	SPR(1000)	STP	nSTM	WPT	UniBi	LOW	74.92	76.60	**76.92**	76.39
(18)	SPR(1000)	STP	STM	WT	UniBi	LOW	75.13	76.50	69.65	**77.13**
(19)	SPR(1000)	STP	STM	WPT	UniBi	SNS	75.97	**77.98**	71.76	77.66
(20)	SPR(12786)	STP	STM	WPT	UniBi	SNS	48.05	**83.24**	74.81	82.30

Table 3. The results of the sparse classification from the »agree3« data set compiled in a table: Accuracy in percent, where the best results per line is written in bold letters

	Used settings (see Table 1)						NB	SVM	ME	Tree
(21)	DNS	STP	STM	WPT	UniBi	LOW	82.71	**83.34**	83.13	82.72
(22)	DNS	STP	STM	WPT	Uni	LOW	82.61	**83.34**	83.55	83.03
(23)	DNS	STP	STM	WPT	Bi	LOW	75.76	**75.97**	75.13	75.65
(24)	DNS	STP	STM	WPT	Tri	LOW	75.13	**75.23**	74.92	75.02
(25)	DNS	STP	STM	WPT	UniBiTri	LOW	82.61	82.61	**82.92**	82.82
(26)	DNS	nEXSTP	STM	WPT	UniBi	LOW	81.34	83.87	**84.51**	81.98
(27)	DNS	nSTP	STM	WPT	UniBi	LOW	79.87	**83.14**	82.50	81.03
(28)	DNS	STP	nSTM	WPT	UniBi	LOW	82.29	**82.67**	82.19	81.24
(29)	DNS	STP	STM	WT	UniBi	LOW	79.87	**80.61**	80.50	79.24
(30)	DNS	STP	STM	WPT	UniBi	SNS	82.61	83.03	**83.34**	82.77

Table 4. The results of the dense classification from the »agree3« data set compiled in a table: Accuracy in percent, where the best results per line is written in bold letters

slightly (4.59 %) increases the accuracy. Leaving the features without stemming (experiment 17) again seems to reduce the accuracy slightly, with the exception of the ME classifier.

The difference between the two tokenizers can be seen in experiments 18 and 4. Here, the results for the WordPunctTokenizer are slightly better. This is probably due to the fact that the emoticons provide information concerning the sentiment. Experiment 19 and 20, compared respectively to experiment 4 and 10, show that leaving the features in upper and lower case has no significant impact on the accuracy at all. That is interesting, since compared to English, upper and lower case in German is quite important for the spelling of a sentence.

Comparing the standard for the dense feature vector (experiment 21, Table 4) to the sparse feature vector (experiment 4) shows a significant improvement (up to 11 % for the ME) in classification accuracy. With this feature vector only the uni-grams (experiment 22–25) seem to perform well, and it does not make any difference if the uni-grams are paired with bi-grams and/or tri-grams or not.

The overall best result (84.51 %) is provided by the ME classifier with removal of the NLTK stopwords (experiment 26). Removing all stopwords (NLTK and extended) decreases the accuracy. Using no stemmer and the WordTokenizer have no positive impact on the resulting accuracy. The effect of leaving the features in upper and lower case is also negligible.

The results from test number 26 with the maximum entropy classifier seems to be the best overall. So this result needs to be investigated further. To do so, we compile the F-measure and its components for every sentiment in Table 5.

Neutral			Positive			Negative		
Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
0.873	0.949	0.909	0.737	0.662	0.688	0.698	0.367	0.475

Table 5. Results from experiment number 26 in greater detail

The data is split into positive, negative and neutral to show how many of the tweets are correctly classified as such. Then for each of these classes the precision, recall and F-measure are listed. The precision shows how many of the tweets classified as a certain class are actually in this class. Recall provides the percentage of all the tweets belonging to this class that was correctly classified as such. Finally, the F-measure is a mixture of precision and recall. The F-measure and recall decrease with the amount of tweets in every class; the neutral class seems to perform best, the negative sentiment performs worst. Maybe this is an indicator, that the ME classifier tends to overfit the data.

As Table 5 shows, all three indicators are especially high for the neutral category. While positive and negative both share similar precision values, the recall for the negative class is especially bad. This is most likely the result of the relatively low number of negative tweets in the corpus. Likewise, the good results for the neutral category can also be explained by the fact, that most tweets in the corpus are considered neutral.

When comparing the classifiers overall, one can see that the SVM and tree classifiers give the best results when using the classical sparse vector, with 83.45 % and 82.82 % (both in experiment 9) accuracy respectively. But when it comes to the dense vectors, the ME classifier gives the best accuracy in this case with 84.51 % (experiment 26) accuracy.

Another crucial characteristic of the text classification is the time it takes to perform a training and test run. Table 6 exemplarily shows some of these values relative to the fastest execution time. The absolute amount of time taken is irrelevant since it depends on the hardware of the used computer. Only experiment 4 and 21 from Table 2 and Table 4 are shown, most of the other execution times are either quite similar or scale predictably, e.g. uni-, bi- and tri-grams take more time than only uni-grams. The dense approach performs with better accuracy and, except for the tree classification, much faster.

	NB	SVM	ME	Tree
(4)	10.34	10.44	68.35	1207.52
(21)	1.08	1.00	14.20	7471.41

Table 6. The train + test time from experiment 4 and 21 from Table 2 and Table 4 in comparison, relative to SVM, experiment 21

6 Discussion

The behavior found here mostly match what has been discovered by Go et al. [2]:

- The best performance of the sparse feature vectors is achieved when the SVM classification is used.
- A combination of n to m-grams leads to a better accuracy than only using n-grams, at least for the sparse feature vector.
- Up to a given point, the amount of used features in the sparse feature vectors lead to a better accuracy.
- The use of preprocessing steps, like stemming and stopword removal, does not seem to have much effect.

Comparing the results of the dense and the sparse feature vectors leads to the assumption, that the classifiers utilize every n-gram of the dense feature vector and, in doing so, build a model of all available data. In the case of the sparse feature vector in contrast only the k most common n-grams may be used, so it ends up with a less accurate model. When the parameter k has been chosen, so that all n-grams are utilized, the SVM and tree classifier performance is comparable to the dense feature vector.

To investigate this assumption further, one must have a closer look at the implementation of the decision tree classifier and the SVM. It is supposed that the tree classifier creates the decision tree iteratively from all the given data, so the model it builds is more complete. The SVM skips all the features, which are labeled as »false« in the feature vector, thus effectively reducing the sparse down to the dense feature vector.

The dense feature vector can indirectly learn about the term frequency, since some n-grams appear more often than others in this feature vector, while every unique n-gram only appears once for the sparse feature vector. This could also lead to a better overall performance of the dense feature vector.

The difference between German and English (e.g. [2]) text in terms of sentiment analysis is surprisingly small. The stopword removal and stemmer must be changed when working with German text. But despite this, the difference is negligible, which leads to similar accuracies. Most ideas, which have been proven to work in English should easily be adaptable to German. A process of sentiment analysis, that performs well on English-language tweets can, with some minor changes, be transferred to German-language tweets.

Compared to [5] the results presented in this paper are better (79.8 % vs. 84.51 %). However, Narr et al. used a different approach, as our test and train data originated in the same corpus (using crossvalidation), whereas Narr et al. used two different sources.

The ambiguousness of the sentiment classification task is shown by the inter-human-agreement: Only in 53.22 % of the cases all three humans agreed on the sentiment, whereas in 95.50 % of all given cases two or more annotators agreed on the sentiment. Compared to our best result of 84.51 % this displays how subjective sentiment classification even for real humans is. This fuzziness seems to provide an upper bound for the classification accuracy.

In comparison to [2,6,7] one must notice that the approach presented in this paper was performed on three different classes instead of two and tweets instead of movie reviews. Despite these differences the performance of our approach is still comparable to most of the results provided by Pang et al. and Go et al.

In addition to this one can compare these results with one of the recent deep learning approaches, e.g. [8]. The best results (85.4 %) Socher et al. gained with a recursive neural network for two classes are about one percent better than our best result for three classes (84.51 %). In addition to this one must notice that Socher et al. used treebank annotations on word level. Since our approach works with the easier to obtain sentence level annotations and produces a three class prediction instead of two classes its performance is on the same level as one of the recent deep learning approaches.

While most of the prepossessing steps have only small influence on the final results the representation of the feature vector impacts the accuracy and the time performance significantly.

7 Future Work

In future experiments, some points could be improved. A larger corpus could be used to gain more reliable results. Another way to improve results would be to use a better method of selecting features for the sparse vector, like information gain or χ^2, instead of just using the k most common words. Also one could reduce the amount of neutral cases within the data set to balance all three cases out and reduce the bias that comes with the large amount of neutral cases. In addition to this other classification algorithms like hidden markov models or deep learning approaches could be used.

Using a bag-of-words approach, structural information is lost. Thus, sentences containing different sentiments in different parts cannot be detected. Employing some structure preserving features, e.g. graphs, seems to be promising.

Finally, it appears that there is some kind of upper bound, regarding to what is accomplishable with classification methods in combination with a bag-of-words approach. To advance above this level (around 82 % to 84 %), a completely different approach, e.g. as Pang et al. described in [6], might be needed. This upper bound of around 84 % seems to be independent from the baseline performances, since Go et al. also reaches this upper bound with a completely different baseline of 65.2 % [2].

Acknowledgments

We are very grateful to Christian Sander for his help with writing the Python software and running parts of the experiments.

References

1. Bütow, F., Lommatzsch, A., Ploch, D.: Creation of a german corpus for internet news sentiment analysis. Tech. rep., Berlin Institute of Technology, AOT (2016)
2. Go, A., Bhayani, R., Huang, L.: Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford 1, 12 (2009)
3. Go, A., Huang, L., Bhayani, R.: Twitter sentiment analysis. Entropy 17 (2009)
4. Kouloumpis, E., Wilson, T., Moore, J.D.: Twitter sentiment analysis: The good the bad and the omg! Icwsm 11, 538–541 (2011)
5. Narr, S., Hülfenhaus, M., Albayrak, S.: Language-independent twitter sentiment analysis. In: Knowledge Discovery and Machine Learning (KDML), LWA (2012)
6. Pang, B., Lee, L.: A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics. ACL '04, Association for Computational Linguistics, Stroudsburg, PA, USA (2004), http://dx.doi.org/10.3115/1218955.1218990
7. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up?: Sentiment classification using machine learning techniques. In: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10. pp. 79–86. EMNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA (2002), http://dx.doi.org/10.3115/1118693.1118704
8. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 1631–1642. Association for Computational Linguistics, Stroudsburg, PA (October 2013)