UNIVERSITÄT OSNABRÜCK

Institut für Informatik

Dissertation

Reliable General Purpose Sentiment Analysis of the Public Twitter Stream

Nils Haldenwang

Juli 2017

Dissertation zur Erlangung des Doktorgrades (Dr. rer. nat.)
des Fachbereichs Mathematik/Informatik
der Universität Osnabrück.

Betreut durch Prof. Dr. Oliver Vornberger

Danksagungen

Allen voran möchte ich mich bei Herrn Oliver Vornberger, der diese Arbeit und mich betreut, besonders bedanken. Durch sein unvergleichliches Talent, junge Menschen für die Informatik zu begeistern, wurde überhaupt erst mein Interesse an dem Fach geweckt. Er hatte stets eine offene Tür und stand mir mit Rat und Tat auf meinem Weg zur Seite. Die Zusammenarbeit und die Zeit hier am Institut für Informatik habe ich immer als sehr angenehm empfunden und würde ohne zu zögern alles wieder genauso machen. Danke dafür!

Mein Interesse am Thema Datenanalyse entsprang der engen Zusammenarbeit mit Nicolas Neubauer im Rahmen meiner Masterarbeit. Er stand mir immer als Mentor und enger Freund zur Seite, dafür meinen herzlichen Dank!

Weiterhin möchte ich Herrn Frank Köster danken, der sich als Zweitgutachter zur Verfügung gestellt hat. Herr Köster war mit seiner Vorlesung "Datamining" ebenfalls der Ursprung der Inspiration zum Thema Datenanalyse, da Nicolas Neubauer als Teilnehmer der Veranstaltung auf diesen Weg geraten ist, auf den er mich dann glücklicherweise mitnehmen konnte.

Die Bewältigung des gewaltigen Implementationsaufwandes der zahlreichen Experimente und Applikationen wäre ohne die ertragreiche Zuarbeit im Rahmen der von mir betreuten Abschlussarbeiten, Hilfskraftverträge und der Masterprojektgruppe DataTwiSt kaum im gegebenen Zeitrahmen möglich gewesen. Daher möchte ich mich hiermit bei allen beteiligten Studenten ganz herzlich für die ertragreiche Zusammenarbeit bedanken: Miriam Beutel, Jan-Hendrik Bolte, Christoph Eichler, Benjamin Graf, Katrin Ihler, Julian Kniephoff, Arnold Kopf, Florian Krampe, Kai Rechtern, Andreas Rinas, Jan-Philipp Schleutker, Magnus Upmann, Falk Wilke und Arne Wilken.

Schließlich gilt mein Dank auch allen Freunden und Kollegen, die mir durch anregende Gespräche, Korrekturen oder einfach mit einem offenen Ohr stets zur Seite standen: Ann-Katrin Becker, Mathias Menninghaus, Jana Meyer und Maren Mikulla. Besonderer Dank gilt meinem guten Freund und Kollegen Friedhelm Hofmeyer, der als Systemadminstrator mit einer Engelsgeduld die technischen Abläufe unterstützt hat.

Meinen Eltern Heide und Edmund Haldenwang, meinem Bruder Jens Haldenwang und meiner Freundin Larissa Nossels möchte ich für die immerwährende Unterstützung, insbesondere in den von Stress geprägten Phasen während der Promotionszeit, meinen herzlichen Dank aussprechen.

Abstract

General purpose Twitter sentiment analysis is a novel field that is closely related to traditional Twitter sentiment analysis but slightly differs in some key aspects. The main difference lies in the fact that the novel approach considers the unfiltered public Twitter stream while most of the previous approaches often applied various filtering steps which are not feasible for many applications. Another goal is to yield more reliable results by only classifying a tweet as *positive* or *negative* if it distinctly consists of the respective sentiment and mark the remaining messages as *uncertain*. Traditional approaches are often not that strict. Within the course of this thesis it could be verified that the novel approach differs significantly from the traditional approach. Moreover, the experimental results indicated that the archetypical approaches could be transferred to the new domain but the related domain data is consistently sub par when compared to high quality in-domain data. Finally, the viability of the best classification algorithm could be qualitatively verified in a real-world setting that was also developed within the course of this thesis.

Zusammenfassung

General Purpose Twitter Sentiment Analysis ist ein neues Forschungsfeld, welches zur traditionellen Stimmungsanalyse von Twitterdaten zwar ähnlich ist, sich in einigen Teilaspekten aber dennoch deutlich unterscheidet. Der Hauptunterschied besteht darin, dass der neue Ansatz den ungefilterten Twitter-Stream als Datengrundlage verwendet, während die bisherigen Ansätze oft sehr starke Filterung vornehmen. Diese Filterung ist für viele Anwendungen nicht realitätsgetreu. Ein weiteres Ziel des neuen Ansatzes besteht darin nur solche Tweets mit den Stimmungen *positiv* und *negativ* zu versehen, welche auch eindeutig so zu klassifizieren sind und alle verbleibenden Nachrichten als *unklar* zu betrachten, um diese bei stimmungsbasierten Analysen unbeachtet lassen zu können. Die traditionellen Ansätze sind oft nicht derartig strikt. Im Verlauf der vorliegenden Arbeit konnte nachgewiesen werden, dass der neue Ansatz sich tatsächlich signifikant vom traditionellen Ansatz unterscheidet. Zwei archetypische Verfahren wurden intensiv untersucht und die Ergebnisse zeigen, dass diese sich gut auf das neue Problem übertragen lassen. Die bisher verwendeten Datensätze hingegen konnten zwar nutzbringend eingesetzt werden, waren aber an das neue Problem angepassten Datensätze klar unterlegen. Schlussendlich konnte die Realitätstauglichkeit des besten gefunden Ansatzes im Rahmen einer realen Anwendung verifiziert werden.

Contents

1 Introduction

Social media has become a popular and widely used tool for communication, expressing ones opinions and connecting with other people. Twitter[1] is one of the most popular social networks with 328 million monthly active users[2]. Twitter can further be characterized as a so called micro blogging platform. While a traditional blog usually consists of longer articles with a posting frequency of once up to several times a week, an active Twitter user usually posts multiple tweets with a character limit of 140 per day. Since the entry barrier is so low, people often tend to react to events in the real world almost instantaneously which opens up the possibilities to harness the publicly available data for various interesting applications. Many of such applications require the reliable classification of the tweet's sentiment. For a multitude of applications such as sales prediction[3], stock market prediction[4] or political use cases[5] it has been shown that practical value can be gained with the sentiment analysis of tweets.

The aforementioned applications are all rather domain specific and do not analyze data that is representative for the unfiltered public Twitter stream. However, there are various applications which require the analysis of the unfiltered public stream. One such application may be the monitoring of the sentiment in a specific region. It is not known beforehand what the people in this region will actually write about, hence, a more general approach is required. Recognizing an increase of negative messages in crisis regions can be useful to get counter measures into place before the situation escalates completely. Another application is the timely discovery of tweeting behaviour that indicates that the user suffers from depression so he can receive help in time.[6] In this case, it also is not known beforehand what the user will tweet about but being able to notice that a person is mainly writing negative messages maybe useful as a warning system. Another application that can benefit from a more general approach is the profiling of certain user accounts with respect to the topics they talk positively and negatively about.[7] Such a profile can be beneficial when important influencers are to be identified for marketing purposes. Finally, the most relevant application for this work is the automatic detection and sentiment analysis of events in real-time.[8] When something happens in the world, news often travel faster through social media than through traditional news channels. Being informed timely and automatically about what is currently happening in the world and how it is perceived by people is of interest for various parties. On the one hand, there is just the usual citizen who is interested in

[1] `https://twitter.com`, last checked on 12.07.2017.
[2] According to `https://about.twitter.com/company`, as of 12.07.2017.
[3] See for example Asur and Huberman (2010), Dijkman et al. (2015).
[4] See for example Bollen et al. (2011), Bernardo et al. (2017).
[5] See for example Diakopoulos and Shamma (2010), Chambers et al. (2015), Smailović et al. (2015).
[6] See for example Wang et al. (2013).
[7] See for example Simsek and Karagoz (2014).
[8] See Petrović et al. (2010), McCreadie et al. (2013) for an example of first story detection on Twitter.

staying up-to-date. On the other hand, there may be governmental or other organisations which have to be notified of certain events such as catastrophes in timely fashion to bring help to the victims and maybe prioritize aspects where especially negative sentiment could be recorded for. Since it also cannot be known beforehand what events will occur, algorithms are needed which are able to deal with the entirety of the public Twitter stream and reliably determine the sentiment of the tweets for later analysis.

One of the most popular datasets used for benchmarking Twitter sentiment analysis approaches is that of the annually held SemEval competition[9]. While being more generic and of higher quality than other datasets, it still is not quite representative for the public twitter stream due to various filtering that was applied beforehand, such as only including tweets about popular topics and tweets which include words from a sentiment lexicon. The class labels used in this dataset were *positive* (sentiment is mainly positive), *negative* (sentiment is mainly negative) and *neutral* (there is no sentiment). Looking at various samples from the public Twitter stream it was noticed by the author, that those classes are not completely feasible for representing the unfiltered stream. If important decisions are to be based upon the analysis, the polarity bearing classes *positive* and *negative* should only be assigned to a message if it is distinctly of the respective sentiment, and not just mostly. Hence, another class is introduced in this work which is called *uncertain* and consists of messages which are *neutral* or where a distinct sentiment cannot be reliably chosen so those messages can be excluded from the analysis. This approach provides more reliable results by being slightly more conservative. Moreover, the unfiltered stream consists of lots of *spam* messages, mainly automatically posted tweets or advertising, which also needs to be dealt with somehow.

Hence, the main goal of this thesis can be summarized as the endeavor to find a suitable algorithm for performing reliable general purpose Twitter sentiment analysis with sufficient quality for real world applications. Since there are no previous results for the novel variation of the problem, the basic state-of-the-art approaches and the available datasets from related domains have to be thoroughly investigated with regard to their transferability to lay down the groundwork for future work in this field.

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of technical details and algorithms which are necessary for comprehension of the later chapters. A chronological review of the history and the state-of-the-art for Twitter sentiment analysis is provided in chapter 3. Chapter 4 consists of the presentation of a large scale experiment for determining the correct preprocessing pipelines for various commonly used features. Chapter 5 starts with the introduction of a high quality dataset for measuring an algorithms' performance for general purpose Twitter sentiment analysis, followed by various experiments which evaluate archetypical approaches from related domains along with some novel ideas. Furthermore, a proof-of-concept implementation of a real-time event detection system with explorative data analysis capabilities is presented in chapter 6. Qualitative evaluations on real-world data are also presented, to solidify that the best performing classification system that could be determined yields practical value. Finally, the results are summarized and concluded in chapter 7.

[9] See Nakov et al. (2013), Rosenthal et al. (2014; 2015), Nakov et al. (2016) for details.

2 Technological and Theoretical Background

In the author's opinion, a dissertation should be as self-contained as possible without being inflated by too many details which are rather distracting to experts of the field. However, a reader who is not familiar with all technical details and algorithms should at least be provided with some basic information that is necessary to follow the rest of the thesis. To cater to both audiences, it was chosen to extract those remarks into a separate chapter so the experts are able to just skip the chapter but the interested reader is able to come back and look up the information as is required. This chapter does not necessarily has to be read from start to end, because the relevant sections are referenced in all the places where they are relevant for comprehension.

2.1 Preprocessing and Vectorizing Text Data

Many commonly used classifiers are based upon statistical methods or numerical computations. Hence, textual input such as tweets has to be somehow transformed into a numerical representation. The following section outlines the dominant approaches for vectorizing text data, namely *manual feature engineering* and *distributed word representations* which can be generated in an unsupvervised manner.

2.1.1 Manual Feature Engineering

Usually, text data is transformed into a vector in a high dimensional euclidean vector space. Feature engineering denotes the process of constructing a transformation, usually in the form of a pipeline of preprocessing steps and feature extractors, which transforms a tweet given as a string into a numerical vector that can be used by various classifiers. Figure 2.1[1] schematically depicts the steps of processing a tweet.

As a first step, the text is usually separated into so called *tokens* which usually correspond to words. Tokenizing the text *"I love New York!"* may result in the token list [`I, love, New, York, !`]. The chosen feature extractors then operate on this list and compute numerical values from the tokens, such as the count of upper case letters or the number of punctuation marks.

Sometimes it makes sense to preprocess the token list before passing it to the feature extractors. Commonly used steps include replacing or removing tokens, such as stop words, punctuation or various entities, replacing words with their infinitive form or downcasing all characters of a token.

[1] I would like to thank Julian Kniephoff for creating this image in the context of the master project group I supervised.

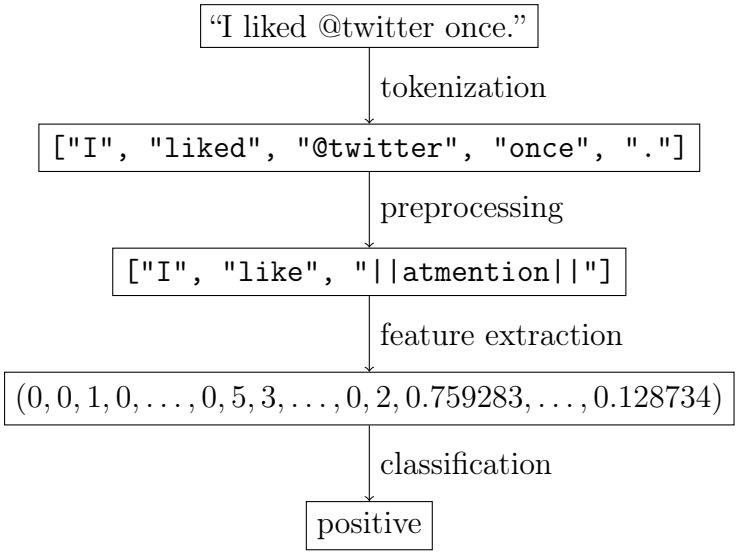


Figure 2.1: Schematic depiction of the whole process of tokenization, preprocessing, feature extraction and classification.

The following sections consist of a detailed introduction of the feature extraction methods and preprocessing techniques which are used throughout this thesis.

Bag-of-Word Models

The canonical approach of representing texts as vectors transforms them by utilizing so called *n-grams*. An n-gram is a contiguous sequence of n items from a given sequence of text. Table 2.1 shows the n-grams for $n \in \{1, 2, 3\}$ of the sentence *The sun is shining today.* Groups of n words are formed for word-level n-grams. The first group begins with the first word and also contains the following $n - 1$ words. After this, the second group is constructed by starting with the second word and taking the following $n - 1$ words. Finally, the described process is continued until the end of the sequence is reached. To put it simple one could say that a window of n words is constructed which moves over the text word by word and extracts a snapshot at each position. The entirety of the snapshots make up the n-gram representation of the text.

Unigrams, $n = 1$	**Bigrams**, $n = 2$	**Trigrams**, $n = 3$
The, sun, is, shining, today	The sun, sun is, is shining, shining today	The sun is, sun is shining, is shining today

Table 2.1: Unigrams, bigrams and trigrams for the sentence “The sun is shining today.”, separated by commas.

Transforming a text into n-grams divides it into an ordered list of tokens, which is not a vector yet. To construct a vector, each token is considered as a feature, which means

each token is mapped to an index of the vector. Given the underlying text corpus' vocabulary V, the size of the vector will be $|V|$. Considering unigrams, this could be tens to hundreds of thousands, as for a reasonably large corpus the number of words will be approximately the number of words in the language of the texts.

Finally, the values of the features have to be computed. These values are often set to the term frequency $TF(w, d)$, which is simply the number of occurrences of the word w in the document d. This model is also referred to as the *bag-of-word* model. Figure 2.2 illustrates this. However, it is often claimed that it is beneficial to just use binary values for term presence in a document instead of its frequency (Pak and Paroubek 2010). There is usually no difference in these two approaches regarding tweets, since the words do not occur more then once anyway due to the short lengths of the messages.

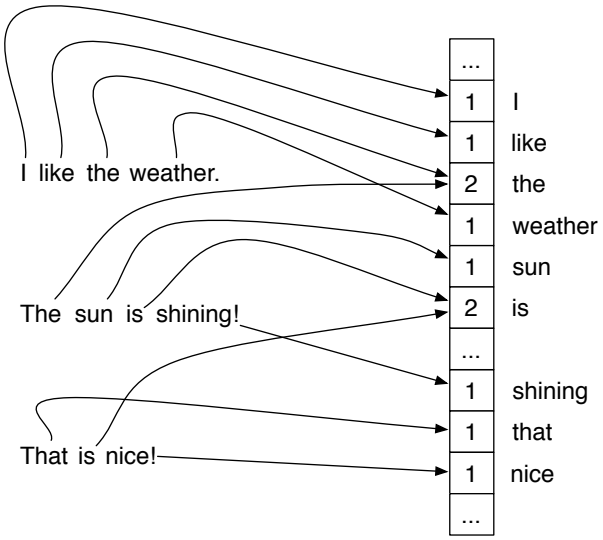


Figure 2.2: Illustration of the unigram bag-of-words vector of a text using term frequency as values.

Considering informal short texts like tweets, the data sparsity increases with higher orders of n-grams. The probability for a bigram to be seen in the training phase is significantly smaller than for an unigram. Hence, bigrams or higher order n-grams are often not suitable to be used as stand-alone features. Thus, most of the time the unigram word model is used. Nevertheless, bag-of-word models with unigram features include some very naive assumptions. First of all, it is assumed that the order of the words is irrelevant, and that the words have no interconnection. Unfortunately, this model is not able to capture negations. Within the context of sentiment analysis this could be very important, because "not happy" obviously is of negative sentiment, which the simple unigram bag-of-words model is not able to capture. Still, it performs quite well for various text classification tasks, despite its naive assumptions (Joachims 2002). Moreover, it is often used as a baseline to compare with in Twitter sentiment analysis and plays an important role as an integral part of multiple algorithms.[2]

[2]See for example Go et al. (2009), Saif et al. (2012a;b), Bakliwal et al. (2012), Mohammad et al. (2013), Zhu et al. (2014).

Sub-word level n-grams (called *subgrams* in the following) are also very popular (Joachims 2002). Some promising results for text classification have first been reported by Neumann and Schmeier (1999). In contrast to word level n-grams the text window does not move wordwise, but characterwise. Consequently, the building blocks of the model are now groups of n characters, and not n words anymore. The string *"computer"*, split up into trigrams ($n = 3$), results in the tokens: *_co, com, omp, put, ute, ter, er_*. The beginning and the end of a word are often marked with an underscore to emphasize that the n-gram did not occur within a word.

One benefit of sub-word level n-grams lies in the fact that they naturally model similar words similarly. Take the words *"computer"* and *"computers"* as an example. Without the necessity of special linguistic analysis, the model captures the similarity of these words, because they have multiple trigrams in common. Therefore, the model would treat them very similarly which obviously is the desired behavior. Furthermore, this representation is language agnostic. In some languages, such as German the correct forms of words are often built in complicated ways. Using sub-word level n-grams, there is no need for language specific linguistic analysis. In contrast, this behavior could also be misleading though. Words like *"computer"* and *"commuter"* also have multiple trigrams in common. In this case the effect is not desired. However, there are further benefits like robustness against spelling mistakes. Especially the informal language of tweets is full of abbreviations and spelling mistakes. Mistakes made by wrong interpretation of these similar words will probably have no significant effect once the training corpus is big enough.

Subgrams are by now a commonly used feature, see for example Mohammad et al. (2013), Zhu et al. (2014), Neubauer (2014).

Preprocessing

Many current methods for Twitter sentiment analysis or even text classification in general include various steps of data preprocessing. One of the most important goals of preprocessing is to enhance the quality of the data by removing noise. Another point is the reduction of the feature space's size, because some methods may struggle with large feature vectors due to limitations in computation time and available memory. In the following the preprocessors are introduced with the names that are used for them throughout the rest of this thesis. Preprocessors are named with common suffixes to indicate their respective action, *Remover* implies the affected tokens are removed without substitution, *Replacer* indicates that the affected tokens are replaced with another token or placeholder and *Annotator* modifies a token by prepending or attaching pre- or suffixes. The behavior of preprocessors without the aforementioned suffixes are explained in more detail in their respective description below.

StopWordRemover One very popular preprocessing technique is stopword removal. Stopwords are words which in general do not carry much meaning or sentiment, for example "the, is, at, which, on". In the field of Twitter sentiment analysis those stopwords are often removed without providing any evidence that they are actually of no value (Pak and Paroubek 2010, Bakliwal et al. 2012, Liu et al. 2012).

One possible drawback of removing the stopwords could be that named entities whose names consist of such stopwords, such as "The Who" or "Take That", could not be recognized anymore. In addition, Saif et al. (2012b) provide evidence that removal of stopwords makes classifiers perform worse. Nevertheless, the reduction in corpus size is reported to be up to 38.3% (Agarwal et al. 2011).

PorterStemReplacer, LemmatizeReplacer Stemming and lemmatization are two very similar preprocessing steps, of which at least one is used in nearly all current methods of text analysis. Stemming is the process of reducing a given inflected word to its stem. The goal is to map similar words to the same stem, which is not necessarily the word's base form. For example, the words "stemmer", "stemmed", "stemming" would all be reduced to "stem". The reasoning behind this reduction is that one wants to capture the sentiment of the general concept of a word and not of all its various inflections. Lemmatization on the other hand takes into account the context of the word and also performs a dictionary lookup. Whereas a stemmer would not be able to reduce the word "better" to "good", a lemmatizer is able to do so. However, lemmatization takes much longer than stemming and may not yield any improvements. Some methods use stemming (Liu et al. 2012), others use sophisticated lemmatization (Bonev et al. 2012) and some use none of them (Saif et al. 2012a).

SlangAcronymExpander Especially in the context of tweets, the language is informal most of the time. People use lots of abbreviations due to the 140 character limit of tweets. For example they often write "thr" instead of "there". Agarwal et al. (2011) suggest an acronym dictionary with more than 5,000 expansions.[3]

MultiCharSpellingCorrector Twitter users tend to spell words intentionally wrong to emphasize them, one example being "looooooove". As the number of repeated letters can be arbitrary, it makes sense to normalize them. In order to be still able to distinguish the emphasized spelling from the correct spelling, the number of repeated letters is often reduced to two (Go et al. 2009, Agarwal et al. 2011, Saif et al. 2012a;b). The misspelled word "looooooove" would become "loove" and so would "looooooooooooooooooove". Saif et al. (2012a) report the reduction of the vocabulary size to be 3.48% on their corpus, Go et al. (2009) achieved 2.77%.

NamedEntityRemover, NamedEntityReplacer Named entity replacement/removal can also make a model more robust. Tweets often contain names of entities, like locations, people or companies. In general, one does not want the model to learn a sentiment towards a certain entity. If, for example, a company had very bad press in the time frame where the training data was collected, the model would always interpret it as negative. Moreover, if the sentiment towards this company shall be tracked, it would probably never change. One possible solution to this is to replace these entities with wild cards, for example the word "London" would be replaced with "||LOCATION||" (Bonev et al. 2012). Other methods ignore nouns in general, since they are of the opinion that nouns do not carry any sentiment at all (Bakliwal et al. 2012). Still, there is no sound evidence regarding this hypothesis yet.

[3]compiled from `http://www.noslang.org`, last checked on 13.07.2017

AtMentionRemover, AtMentionReplacer Tweets contain specific entities such as mentions of other users, starting with @. It is a common practice to also replace those with wild cards, such as "||USERNAME||" (Go et al. 2009, Pak and Paroubek 2010, Liu et al. 2012, Saif et al. 2012a). It is assumed that the exact user name is of no consequence for the tweets' sentiment but one may be interested in learning structure associated with a user name. The largest benefit is the drastic reduction in corpus size. Go et al. (2009) report a reduction in vocabulary size by 43.42%, Saif et al. (2012a) report 28.58%.

URLRemover, URLReplacer The removal of URLs is analogous to the handling of user names but the wild card used is "||URL||". While still significantly reducing the corpus size, the reduction is usually not as strong as for user names, Go et al. (2009) report a reduction of 9.41%, the vocabulary of Saif et al. (2012a) decrease by a relative amount of 2.91%.

PartOfSpeechAnnotator, PartOfSpeechReplacer Part-of-Speech Tagging, also often called grammatical tagging or word-category disambiguation, refers to the process of tagging a word in a text as a certain part of speech, depending on its original definition and its context. Most people learn to identify nouns, verbs, adverbs and adjectives at school, which are just a small subset of what current POS taggers are able to tag. Figure 2.3 shows a screenshot taken from an online POS tagger for the sentence "Oh man, I really like this new smartphone!".

Figure 2.3: The sentence "Oh man, I really like this new smartphone!", POS tagged by an online demo of the University of Illinois (`http://cogcomp.cs.illinois.edu/demo/pos`, also see Roth and Zelenko (1998)).

The used POS tagger is able to identify a total of 47 tags, some are shown in figure 2.3. First of all, the tag *UH* at the word "Oh" means interjection, a word which expresses an emotion but is also often used to fill pauses. Besides this, the tagger is able to identify different types of nouns. Looking at the words "man" and "smartphone" one notices they are both tagged with *NN*, which means singular noun. The word "I", which is also a noun, is tagged with *PRP*, meaning personal pronoun. The complete list of tags is available on the project's website.[4]

However, POS taggers for formal texts usually do not work for informal language such as that on twitter. Saif et al. (2012b) suggest to use a tagger called *TweetNLP*, which was developed at the Carnegie Mellon University specifically for tagging tweets.[5]

[4] `http://cogcomp.cs.illinois.edu/demo/pos`, last checked on 13.07.2017.

[5] `http://www.ark.cs.cmu.edu/TweetNLP/`, last checked on 13.07.2017. For details on its construction and progress please see Gimpel et al. (2011), Owoputi et al. (2013).

The tagger is able to handle the informal language of tweets very well. There are even special POS tags for tweet specific tokens like emoticons, hashtags, mentions and URLs.

ikr	smh	he	asked	fir	yo	last	name	so	he	can	add	u	on	fb	lolol
!	G	O	V	P	D	A	N	P	O	V	V	O	P	∧	!

Table 2.2: Example of a tweet tagged with TweetNLP, taken from Owoputi et al. (2013).

Figure 2.2 shows a rather extreme example of nonstandard orthography, abbreviation and misspelling. Its meaning is basically "He asked for your last name so he can add you on Facebook.". The tagset differs from the one of TreeTagger, which uses the Penn Treebank-style tagset (PTB). A complete list can be found in the appendix of Owoputi et al. (2013). However, recently a PTB-style tagset was released for TweetNLP, in case one wishes to use that instead.[6]

The tagging works remarkably well. For example the abbreviation "ikr", which means "I know, right?" is correctly tagged as an interjection (!). Besides this, the phrase "yo", meaning "yours", is recognized as a possessive pronoun (D). As a final example, it even recognized "fb" as an abbreviation for "Facebook" and tags it as a proper noun (∧).

NegationAnnotator Annotating negated tokens is a common practice in sentiment analysis tasks[7]. The main idea is that words such as "good" are connoted with a strong positive sentiment but if they are used in conjunction with a negation completely invert the sentiment, for example: "The new android phone is not good.". Most approaches follow the simple but presumably effective procedure suggested by Pang et al. (2002). A negated context is defined as a part of the tweet that starts with a negation such as "not or n't". The end of the context is usually chosen as the next punctuation mark or the end of the tweet. All tokens within the negated context are then annotated to be negated by adding a suffix or postfix such as *_NEG_* or *!*. Annotating the example from above in this manner would result in the token list `[the, new, android, phone, NEG_good]`.

DowncasingPreprocessor Downcasing of all characters within a tweet is performed with the underlying assumption that there is no consistent and meaningful usage of upper case letters in the informal language used within social networks, such as Twitter. Even though writing in all upper case letters is some times considered *shouting* in many internet communities, it probably also is not used consistently for this purpose. Moreover, authors of tweets tend to use seemingly random upper case letters within words for styling there messages, for example: "mE goTTa eXeCuTe dAt sTuFF". Those tokens do not carry any extra meaning compared to their completely lower cased counterparts. Hence, downcasing is used to normalize such tokens to increase the density of information and reduce the dimension of the feature space.

[6] See `http://www.ark.cs.cmu.edu/TweetNLP/` for more information.
[7] See for example Pang et al. (2002), Agarwal et al. (2011), Mohammad et al. (2013).

PositionAnnotator Pang et al. (2002) used position annotation as a preprocessing step for the sentiment analysis of movie reviews. Words are annotated with their position within the text, since it was assumed that sentiment expressed in later parts of a review is more important. Transferred to short messages such as tweets, it makes sense to annotate each token with the third in which it occurred; the message "I love you. Not." becomes the token list `[I-1, love-2, you-3, not-3]`[8].

Word Cluster Features

Word cluster features were first - to the best of the author's knowledge - proposed to be used for Twitter sentiment analysis by Mohammad et al. (2013).

Owoputi et al. (2013) argue that previous work has established unsupervised word clusters as a method for improving various supervised natural language processing tasks[9]. To be used as improvement for their POS tagging system, Owoputi et al. (2013) created 1,000 unsupervised hierarchical Brown clusters as proposed in Brown et al. (1992) based on a tweet corpus of 56 million tweets consisting of 847 million distinct tokens of which only those have been used that occurred more than 40 times (216,856). Table 2.3 shows some example clusters.

Cluster	Top words (by frequency)
1	lmao lmfao lmaoo lmaooo hahahahaha lool ctfu rofl lool lmfaoo lmfaooo lmaoooo lmbo lololol
2	haha hahahaha hehe hahahahah hahahah aha hehehe ahahah hah hahahah kk hahaa ahah
3	yes yep yup nope yess yesss yessss ofcourse yeap likewise yepp yesh yw yuup yus
4	gonna gunna gona gna guna gnna ganna qonna gonnna gana qunna gonne goona

Table 2.3: Example word clusters from Owoputi et al. (2013).

To use the clusters as features, Mohammad et al. (2013) suggest to create a binary vector $c \in \{0,1\}^{1000}$ with each element $c_i, i \in \{1,2,\ldots,1000\}$ indicating whether the tweet to be represented by c includes a token from word cluster i. If a token from cluster i is included, $c_i = 1$ and $c_i = 0$ otherwise. Since each cluster i represents a group of words the hope is to condense the information. Consider for example cluster 1 from table 2.3. All tokens in the cluster are an expression of laughing which is commonly used in informal texts coming from social networks. There are completely different expressions present such as "lool"(laughing out loud), "rofl" (rolling on floor laughing),

[8] The message uses a common construct to express something ironic. The postponed "not" further emphasizes the irony. This example was taken from Neubauer (2014).

[9] See for example Koo et al. (2008), Täckström et al. (2012), Turian et al. (2010).

"lmao"(laughing my ass off) but also different variations in spelling for the various expressions, such as "lool" and "lolol". Basically, it is not relevant which expression in its respective spelling variation is used but rather that there is an expression of laughing present in the tweet.

Lexicon Features

A *sentiment lexicon* associates tokens - usually words - with a sentiment score. The intervals of the score vary from one lexicon to another. Most of the sentiment lexicons have in common that a positive score indicates positive sentiment while a negative score indicates negative sentiment. Moreover, the degree of sentiment is indicated by the absolute value of the score. The word *good* may be associated with a score of 3.0 while the word *awesome* may result in a score of 5.0.

To harness an existing lexicon for the generation of features from a tweet Mohammad et al. (2013) suggested to compute the following values for each token w and polarity $p \in \{\text{positive}, \text{negative}\}$:

- total count of tokens in the tweet with $score(w, p) > 0$
- total score $= \sum_{w \in tweet} score(w, p)$
- the maximal score $= max_{w \in tweet} score(w, p)$
- the score of the last token in the tweet with $score(w, p) > 0$

Some popular examples for commonly used manually created sentiment lexicons are the NRC emotion lexicon (Mohammad and Turney 2010), the MPQA lexicon (Wiebe et al. 2005) and the Bing Liu lexicon (Hu and Liu 2004). All of the aforementioned lexicons are created by manually annotating the words or directly assigning scores. The obvious drawback is that creating such resources manually is costly and time-consuming. Moreover, they are not exactly suited for usage in conjunction with the informal language in tweets, since the lexicons were created with formal texts in mind.

To overcome the aforementioned drawbacks, Mohammad et al. (2013) introduced the idea of creating a sentiment lexicon from a twitter corpus in a semi-supervised manner. They labelled tweets according to sentiment bearing hashtags occurring within the messages and computed the pointwise mutual information as a score for the lexicon[10]. Moreover, the authors are not only creating a lexicon for unigrams, but also for various other bag-of-word models in the same manner. About a year later it was discovered that negation influences positive and negative phrases in different intensities and the incorporation of this knowledge into the lexicon creation process was introduced[11] (Zhu et al. 2014).

A multitude of manually and automatically created sentiment lexicons are publicly available and can be used to incorporate this large knowledge base as features by computing simple scores as mentioned above.

[10] See section 3.2.1 for further details.
[11] See section 3.3.4 for further details.

2.1.2 Distributed Word Representations

Distributed word representations provide an alternative to bag-of-word models. The basic idea is described in the following.

Bag-of-word models as introduced in section 2.1.1 suffer from certain drawbacks. Each word in the text that is to be represented by the vector can only influence one (for unigrams) or very few (for higher order n-grams) components of the vector. Hence, the resulting vector space is very large, its dimension at least equals the number of words in the vocabulary. For higher order n-grams the dimension increases drastically. Only a very small subset of such vectors represents actual texts. Not only are the vectors themselves *sparse*, but the vector space itself is sparsely populated by the data.

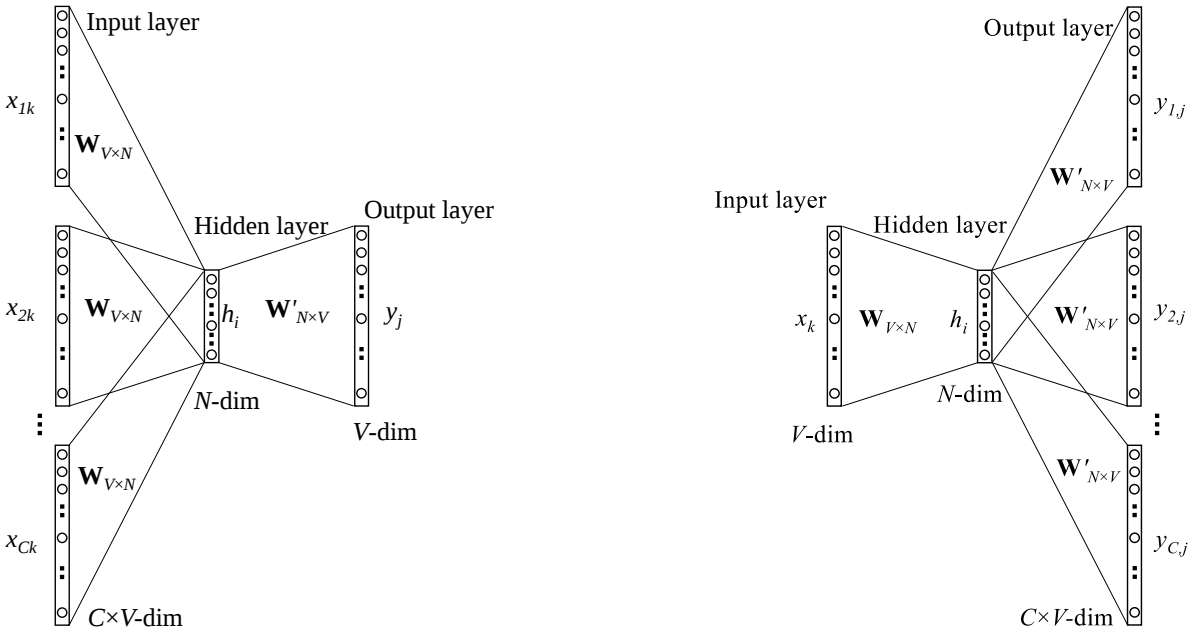
While this is not a negative property in general[12] it still can lead to memory issues. While those can sometimes be rectified with *feature selection* or smart implementations of sparse vectors, the computation time for the classification algorithms often remains costly. Moreover, bag-of-word representations lack any type of semantic information. The distance between the vectors for the word "good", its antonym "bad" and the completely unrelated word "potato" are the same.

To rectify those problems each word is projected into a vector space of relatively low dimension. Commonly used dimensions are in the range of 50-300 as opposed to hundreds of thousands for bag-of-word models. Moreover, the vectors and the vector space itself are generally dense; each component has a meaning for every represented word. One can think of this as a turnaround of the bag-of-word model. For bag-of-word models, the vocabulary directly defines the vector space and the meaning of all the vector components. For distributed representations, the meaning of the components does not have to be set beforehand. The words are embedded into a given vector space where each dimension carries some meaning for the represented word. Hence, distributed representations are often called *word embeddings*. Furthermore, when constructing the vectors it is frequently desired that they include significant semantic information about the words they are representing.

Creation of distributed word representations is commonly based upon the *distributional hypothesis*, which basically makes the assumption that semantically similar words occur in similar contexts within texts (Harris 1954, Sahlgren 2008). Making use of the large text corpora that are freely available nowadays, one can extract this information with statistical methods or machine learning approaches.

A common algorithm to create word embeddings is *word2vec*. The method is described in detail in Mikolov et al. (2013b). Basically, an artificial neural network is trained to predict a word based on its context or the context when given the word. The context of a word is defined as a given number of tokens that occur directly before or after the word in the given text. In the following the two approaches are described more formally:

[12] Joachims (1998) attributes the linear separability of text data to this property.



(a) *CBOW* architecture **(b)** *skip-gram* architecture

Figure 2.4: Schematic representation of the two most common architectures to create word embeddings with neural networks, taken from Rong (2014).

Continuous bag-of-words The input for the CBOW model is a sequence of words where one word in the middle has been deleted. The prediction task the network is then trained to do is to fill in the missing word in the given context.

Continuous skip-grams For the skip-gram model, the task is inverted compared to CBOW. The input is a single word and the network is then trained to predict the surrounding context.

A schematic representation of both models is shown in 2.4. In the following, the CBOW model is used as an example to further illustrate the process of word embedding generation, the approach for the skip-gram model is analogous.

To generate useful word embeddings, a large corpus of example texts is needed.[13] From this corpus all possible $(C + 1)$-tuples are extracted and the token in the middle is deleted to create C-tuples $(w_1, \ldots, w_C)$. Hence, C denotes the context size.

The input for the network are one-hot vectors which are similar to those of the bag-of-word models. All words in the corpus make up the vocabulary and are assigned to a fix position in the one-hot vectors. The presence of a word is denoted by the corresponding vector element to be set to 1. The size of this vocabulary is referred to as V.

For each possible word $1 \leq k \leq V$ and each position $1 \leq c \leq C$ in the context, there is an input neuron x_{ck} whose activation function is 1 if and only if the word on position c in the current context is word k from the vocabulary and 0 otherwise.

All those input neurons are fully connected with the internal layer of size N. The weights of those connections are stored in a $(V \times N)$ matrix W. Note that the weight

[13] Mikolov et al. (2013a) used corpora with hundreds of millions of texts.

of the connection from an input neuron x_{ck} to a hidden neuron h_i with $1 \leq i \leq N$ is independent of the context position c and is given as $\frac{W_{ki}}{C}$ where W_{ki} is the entry in matrix W at row k and column i.

The output layer consists of V neurons y_j which represent all words of the vocabulary. The weights for the connections of the internal neurons h_i and the output neurons y_j are kept in the $(N \times V)$ matrix W'. As an activation function, the *softmax* function $\sigma \colon \mathbb{R}^V \to \mathbb{R}^V$:

$$\sigma_w(x) = \frac{e^{x_w}}{\sum_{i=1}^{V} e^{x_i}} \tag{2.1}$$

with $1 \leq w \leq V$ is used. The function enforces all output values to be in the interval $[0, 1]$ and sum up to 1. Hence, the output values y_j can be interpreted as probabilities of how probable the networks thinks the word j to be the missing word in the context. With an appropriate target function the network can then be trained with common approaches, for further details see Mikolov et al. (2013a).

The word embeddings of dimension N can now be found in the rows of the matrix W or the columns of matrix W'. Garten et al. (2015) investigated and discussed various combinations of both.

Table 2.4 illustrates the semantic information that is encoded in the resulting word embeddings. The semantic similarity is even reflected in the algebraic structure of the N dimensional vector space:

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \simeq v_{\text{queen}} \tag{2.2}$$

where v_w denotes the word embedding vector for word w and the relation $u \simeq v$ of an arbitrary vector $u \in \mathbb{R}^N$ and a word embedding v is to be interpreted as "v has the smallest distance to u out of all generated word embeddings".

The relationship of the word embeddings shown in equation 2.2 is quite remarkable. Without any additional input with regard to semantics, the system was able to capture that "king" relates to "queen" the same way "man" relates to "woman". Moreover, equation 2.2 can be interpreted as "Take away from king what defines a man and add what defines a woman and you get a queen.".

The actual algorithm to create word embeddings that was used in this work includes some extensions to the aforementioned method, see Mikolov et al. (2013b) for further details. Some of the additions will be highlighted in the following, where the parameters used in this work are described[14]:

- About 33 million tweets consisting of $V = 624,015$ unique tokens collected by Neubauer (2014) were used as corpus.

[14] Whenever word embeddings are mentioned in the rest of the thesis, these are the parameters that were used unless stated otherwise.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Table 2.4: Analogy examples that are encoded in the linear structure of word embeddings. Each row can be interpreted analogously to equation (2.2) for the analogy "man : woman :: king : queen". The examples are taken from Mikolov et al. (2013a).

- Minimal preprocessing was applied to reduce the most prominent sources of noise:[15]
  - `AtMentionReplacer`
  - `URLReplacer`
  - `NonWordRemover`
- The dimension of the word embeddings was $N = 100$.
- Two training iterations have been performed.
- The context size was $C = 5$.
- To remedy the imbalance of the influence of very frequent words compared to infrequent words, words with a relative frequency higher than 10^{-5} are ignored with a certain probability.
- *Negative sampling* was applied to prevent overfitting. Additionally to each word/-context pair that was created as described above, k random pairs are generated. The network is then trained to output all zeros for the random contexts. The value $k = 5$ was used here.

The aforementioned choices were made based on the recommendations from Mikolov et al. (2013b). To handle the actual computation, the *gensim*[16] library was used. All parameters not explicitly mentioned here have been set to the default of gensim.

[15] See section 2.1.1 for a detailed description of the preprocessing steps.
[16] See Řehůřek and Sojka (2010).

2.2 Classification Algorithms

The basic ideas of the most relevant classification algorithms used in this work are outlined and enriched with references for further reading in the following sections.

2.2.1 Naive Bayes Classification

Bayesian classifiers are statistical classifiers which are able to predict the probability of a given sample to belong to a particular class. The simplest Bayesian classifier, known as *Naive Bayes Classifier* (NBC), is comparable in performance with *Decision Trees*, *Neural Networks* (Han et al. 2006) and, using various smoothing techniques (Yuan et al. 2012), even with *Support Vector Machines* (SVM).

Bayes Theorem

The following explanation is closely based upon Han et al. (2006), but was slightly simplified and shortened.

Let X be a data sample with unknown class label and let H be a hypothesis, such as that the data sample X belongs to a specified class C. The classification problem is to determine $P(H|X)$, the probability that given a sample X the hypothesis H holds.

$P(H|X)$ is called *posterior probability* of H conditioned on X. Within the domain of Twitter Sentiment Analysis the data samples consist of tweets, their canonical features being the words. Given X contains the words "sad" and "bad" and H is the hypothesis that the tweet is of negative sentiment. Then $P(H|X)$ reflects the confidence that X has a negative sentiment, given we know it includes the words "sad" and "bad".

In contrast, $P(H)$ is the *prior probability* of H, the probability of the hypothesis holding for any given sample X. For example, this is the probability that any given tweet has a negative sentiment, regardless of which words it contains. Note the prior probabilities independence of X, whereas the posterior probability is based on additional information, such as background knowledge.

Similarly, $P(X|H)$ is the posterior probability of X conditioned on H. In the example this would be the probability that X contains the words "sad" and "bad", given we know its sentiment is negative. $P(X)$ is the prior probability of X, in the example it is the probability of a tweet containing the words "sad" and "bad".

The question is: How can these probabilities be estimated? First of all, $P(X)$, $P(H)$ and $P(X|H)$ may be estimated from the given data. To finally calculate the posterior probability from these probabilities one can harness the *Bayes Theorem*:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \ . \tag{2.3}$$

Classification

This section is also based on Han et al. (2006) and slightly adapted to fit the needs of this thesis.

Each data sample is represented by an n-dimensional vector $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the sample from n attributes $A_1, A_2, \ldots, A_n$.

Let $C_1, C_2, \ldots, C_m$ be m classes to which an unknown given sample X can be assigned. The classifier will predict X to belong to the class C_i having the highest posterior probability, conditioned on X. That is, the naive Bayes classifier assigns an unknown sample X to class C_i if and only if

$$P(C_i|X) > P(C_j|X) \text{ for } 1 \leq j \leq m,\ j \neq i\ . \tag{2.4}$$

Thus, we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. It can be computed using the Bayes Theorem 2.3:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}\ . \tag{2.5}$$

Owing to the fact that $P(X)$ is constant for all classes, $P(X|C_i)P(C_i)$ needs to be maximized. One problem which can occur is a lack of knowledge about the class prior probabilities. Therefore, it is commonly assumed that the classes' occurrences are equally probable: $P(C_1) = P(C_2) = \cdots = P(C_m)$. One would maximize $P(X|C_i)$. Otherwise $P(X|C_i)P(C_i)$ would be maximized. Nevertheless, the class prior probability may be estimated by

$$P(C_i) = \frac{s_i}{s}\ , \tag{2.6}$$

where s_i is the number of training samples of class C_i and s is the total number of training samples.

Text data can contain a nearly infinite number of attributes because there is no limit for forming words, especially due to the excessive use of slang and abbreviations on microblogging platforms. That is why it would be extremely computational expensive, or even impossible, to compute $P(X|C_i)$. Consequently, the computation has to be reduced somehow. Simplifying the computation can be done by making the naive assumption of *class conditional independence*. This assumption presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample. That is, there are no dependence relationships among the attributes. Thus, $P(X|C_i)$ can be computed like this:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)\ . \tag{2.7}$$

The computation of $P(x_k|C_i)$ can be done with the *maximum likelihood estimation*

$$P(x_k|C_i) = \frac{s_{ik}}{s_i} \ , \tag{2.8}$$

where s_{ik} is the number of training samples of class C_i having the value x_k for A_k, and s_i is the number of training samples belonging to C_i.

When dealing with text data, especially short texts like tweets, one would not consider the number of occurrences of a word, but just its presence. Thus, the maximum likelihood estimation is often (Joachims 2002, Saif et al. 2012a;b) formulated as

$$P(w|C_i) = \frac{TF(w, C_i)}{\sum_{w' \in V} TF(w', C_i)} \ , \tag{2.9}$$

where $TF(w, C_i)$ is the occurrence frequency of word w in documents of class C_i, and V is the vocabulary of the underlying text corpus.

In order to classify an unknown sample X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. Sample X is then assigned to the class C_i if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \text{ for } 1 \leq j \leq m, \ j \neq i \ . \tag{2.10}$$

Simply put, it is assigned to the class C_i for which $P(X|C_i)P(C_i)$ is the maximum:

$$\text{classify}(X) = \arg\max_{C_i} P(X|C_i)P(C_i) \ . \tag{2.11}$$

For text classification problems, using equations 2.7 and 2.9, the classification function can be formulated as:

$$\text{classify}(X) = \arg\max_{C_i} \ P(C_i) \ \prod_{k=1}^{n} \frac{TF(w_k, C_i)}{\sum_{w' \in V} TF(w', C_i)} \ , \tag{2.12}$$

with n being the number of words in X.

Since the classification function is basically a product of many small numbers, the probability is often transformed to the so called *log likelihood*, to reduce floating point errors while computing the results (Pak and Paroubek 2010, Bonev et al. 2012):

$$\text{classify}(X) = \arg\max_{C_i} \left(\log P(C_i) \ + \ \sum_{k=1}^{n} \log \frac{TF(w_k, C_i)}{\sum_{w' \in V} TF(w', C_i)} \right) . \tag{2.13}$$

Dealing with Unknown Features: Smoothing Techniques

When the language is informal, such as that in tweets, there are many unknown words, which are not covered by training data. Furthermore, the maximum likelihood estimation for $P(w|C_i)$ (see equation 2.9) cannot be computed because the divisor would be zero. Hence, the model would assign unknown words a zero probability for all of the classes, which is probably not true. Various smoothing techniques have been introduced to deal with this problem by estimating a probability for unknown words.

Zhai and Lafferty (2004) summarize: "In general, smoothing methods discount the probabilities of words seen in the text and assign the extra probability mass to the unseen words according to some fallback model." As their field of research was information retrieval, they exploited the collection language model as fallback. For the purpose of twitter sentiment analysis, the fallback model may be exchanged, but for now the explanation of the general principle will stick to the definitions of Zhai and Lafferty (2004). The nomenclature is slightly adapted to fit the previous explanations.

Chen and Goodman (1996) assume the general form of a smoothed model to be the following:

$$P(w|C_i) = \begin{cases} P_s(w|C_i), & \text{if word } w \text{ is seen} \\ \alpha_d P(w|M), & \text{otherwise} \end{cases} . \tag{2.14}$$

In this equation $P_s(w|C_i)$ is the smoothed probability of a seen word, $P(w|M)$ is the collection language model and α_d is a coefficient controlling the probability mass assigned to unseen words, so that all probabilities sum up to one. Given $P_s(w|C_i)$, α_d must have the form:

$$\alpha_d = \frac{1 - \sum_{w \in V: TF(w,C_i)>0} P_s(w|C_i)}{1 - \sum_{w \in V: TF(w,C_i)>0} P(w|M)} . \tag{2.15}$$

Hence, the essential difference of smoothing methods is the choice of $P_s(w|C_i)$.

The easiest and widely used smoothing method coming to mind is called *Laplace smoothing*. It was suggested by Vapnik (1982) and its idea is as simple as adding an extra count to every word. Even though the idea is not complicated, this technique works well in practice (Joachims 1996; 2002). Applied to the maximum likelihood estimation for text classification (see equation 2.9), the new estimator looks like this:

$$P_L(w|C_i) = \frac{1 + TF(w, C_i)}{|V| + \sum_{w' \in V} TF(w', C_i)} . \tag{2.16}$$

Yuan et al. (2012) argue that just adding one to the occurrence frequency of the word just adds noise, to which classes containing few training data samples are very sensitive. However, as this is not the case for twitter sentiment classification, Laplace smoothing can still be used. For instance, it was used as baseline in conjunction with an unigram language model by Saif et al. (2012b;a).

2.2.2 Support Vector Machines

This section provides an introduction to *Support Vector Machines* (SVMs) and is closely based upon Joachims (2002, chapter 3). SVMs are non-probabilistic linear binary classifiers, which can be used for classification and regression analysis. They are able to handle large feature spaces reasonably well. Firstly, the history of SVMs is presented shortly, followed by an illustration of the various types of SVMs. While the standard SVM is a linear classifier, non-linear problems also can be handled using a so called *kernel trick*, which is illustrated as well. The chapter concludes with a discussion on currently available SVM implementations.

Origins and Basic Idea

Support Vector Machines were developed based on the *Structural Risk Minimization* principle (Vapnik 1982, Cortes and Vapnik 1995, Vapnik 1998). The idea is to find a hypothesis h from a hypothesis space H, for which the lowest error probability $Err(h)$ can be guaranteed for a given sample S:

$$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \qquad \vec{x}_i \in \mathbb{R}^n,\ y_i \in [-1, +1]\ , \tag{2.17}$$

where $\vec{x}_i$ denotes a feature vector and y_i the class label. The true error of a hypothesis h is connected with the error $Err_{train}(h)$ of h on the training set and the complexity of h by the following upper bound (Vapnik 1998):

$$Err(h) \leq Err_{train}(h) + O\left(\frac{d\ ln(\frac{n}{d}) - ln(\eta)}{n}\right)\ . \tag{2.18}$$

The probability of the bound holding is at least $1 - \eta$. Furthermore, d denotes the so called *VC-dimension* (Vapnik 1998), which indicates the expressiveness of the hypothesis space H. Equation 2.18 reflects a trade-off between complexity of the hypothesis space and the training error. On the one hand, a simple hypothesis space with a small VC-dimension will probably not contain good approximation functions. Thus, the training error, along with the true error, will be large. On the other hand, a very large hypothesis space (large VC-dimension) will lead to a smaller training error, but will also increase the upper bound due to its linear influence in the right hand side term of equation 2.18.

Therefore, when the hypothesis space has a high VC-dimension, the hypothesis with a very low training error may just fit the training data without proper generalization. This results in poor performance when predicting unknown examples. In general, such behavior of machine learning algorithms is called *overfitting*. Hence, it is crucial to pick a hypothesis space with correct complexity.

In Structural Risk Minimization, the prevention of overfitting is achieved by nesting hypothesis spaces H_i in a way that their respective VC-dimension d_i increases:

$$H_1 \subset H_2 \subset H_3 \subset \cdots \subset H_i \subset \ldots \quad \text{and} \quad \forall i : d_i \leq d_{i+1} \ . \tag{2.19}$$

This structure has to be defined before analyzing the training data. The problem to be solved is to find an index i^* for which the training error is minimal.

The question is: How can those structures be found in practice?

In Structural Risk Minimization, linear threshold functions with N features are created, resulting in the function's VC-complexity being $N + 1$. Given the features as a ranked list, using the first feature will have a VC-dimension of two, using the first two features will have a VC-dimension of three and so on. For very large feature spaces, as it is the case in text classification, this is not practical. Moreover, it is not clear how to rank the features.

Support Vector Machines learn linear threshold functions of the type:

$$h(\vec{x}) = sign\{\vec{w} \cdot \vec{x} + b\} = \begin{cases} +1, & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.20}$$

These functions correspond to a hyperplane within the feature space. This hyperplane is described by $\vec{w}$, being the hyperplanes normal vector, and b, being the offset from the origin along this normal vector. All vectors $\vec{x}$, which satisfy equation $\vec{w} \cdot \vec{x} + b = 0$, lie within the hyperplane. Hence, classification of an example $\vec{x}$ with $h(\vec{x})$ basically is done by determining on which side of the hyperplane it lies. Vapnik (1998) showed that the VC-dimension of Support Vector Machines is independent of the number of features, but is bound by the margin δ (see the following section 2.2.2 for an explanation of δ). Vapnik (1998) also showed that the VC-dimension becomes smaller the larger the margin δ is. While this property does not guarantee good performance, it guarantees that SVMs do not *necessarily* fail, meaning they are able to perform well for high dimensional classification tasks with a reasonable VC-dimension. For further details see Joachims (2002) and Vapnik (1998).

Linear Hard-Margin SVMs

Let the training samples be tuples $(\vec{x}_i, y_i)$ with $\vec{x}_i$ denoting the vector of feature values and $y_i \in \{-1, +1\}$ denoting the class labels. For simplicity it is assumed that the data is linearly separable, meaning it can be divided by at least one hyperplane h'. Thus, a weight vector $\vec{w}'$ and a threshold b' exist, such that all positive examples are on one side of the hyperplane and the negative examples are on the other side. This is equivalent to:

$$\forall(\vec{x}_i, y_i) : \quad y_i(\vec{w}' \cdot \vec{x}_i' + b') > 0 \ . \tag{2.21}$$

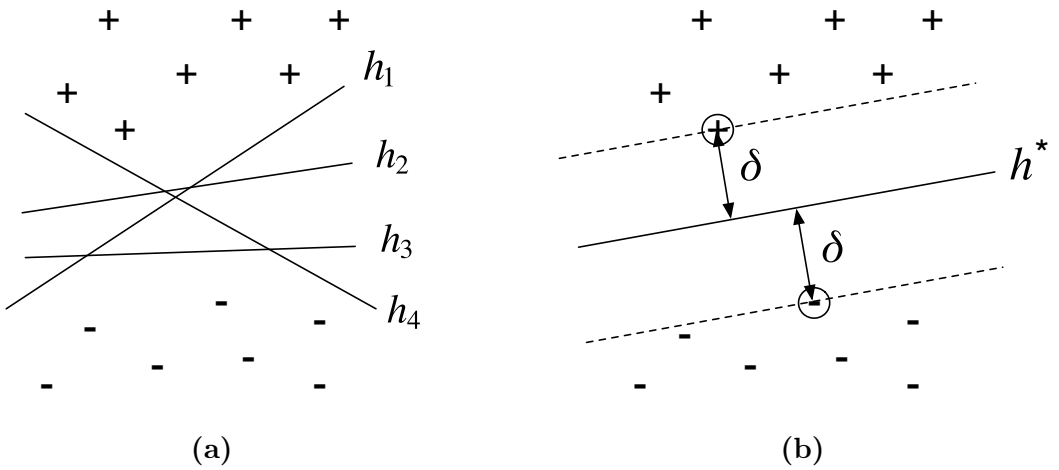


Figure 2.5: Example of a two dimensional binary classification problem. Positive examples are marked by + and negative ones by –. The left figure (a) shows that many hyperplanes separate the training samples without error. Support Vector Machines find the hyperplane h^*, which separates the training examples with maximum margin δ, as shown in the right figure (b). The examples closest to the hyperplane are called *support vectors* (marked with circles). Also see Joachims (2002).

As shown in figure 2.5a there can be an arbitrarily large number of hyperplanes separating the classes without errors. From these the Support Vector Machine chooses the hyperplane h^* with the largest margin δ, as shown in figure 2.5b. Training samples closest to the hyperplane, the distance to it being exactly δ, are called *Support Vectors*. They are marked with circles.

To find the hyperplane h^* with the maximum margin one has to solve the following optimization problem:

Optimization Problem 1 (Hard-Margin SVM (PRIMAL))

$$minimize: \quad V(\vec{w}, b) = \frac{1}{2}\vec{w} \cdot \vec{w} \tag{2.22}$$

$$subject\ to: \quad \forall_{i=1}^{n} : y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \ . \tag{2.23}$$

Equation 2.23 formalizes the condition that every example has to be on the correct side of the hyperplane. Unlike in equation 2.21 the inequalities' right hand side is now one and not zero anymore. This enforces a certain margin δ. As the weight vector $\vec{w}$ also is the normal vector of the hyperplane, it is easy to verify that $\delta = \frac{1}{||\vec{w}||}$ with $||\vec{w}||$ being the L_2-norm of the vector $\vec{w}$. Hence, by minimizing $\vec{w} \cdot \vec{w}$ the margin δ is maximized. The hyperplane h^* is described by $\vec{w}$ and b, which are the solution of the optimization problem.

As this optimization problem is numerically hard to solve, it is often transformed to its Wolfe dual, an equivalent problem having the same solution, which is commonly solved in practice. For further details see Joachims (2002).

Soft-Margin SVMs

The Linear Hard-Margin SVM suffers from the disadvantage, that its training fails if the data is not linearly separable. In this case, there will be no feasible solution to Optimization Problem 1. Although most text-classification problems are linearly separable (Joachims 2002), it may still be beneficial to allow a certain number of errors in training. To overcome this issue, Cortes and Vapnik (1995) developed the *Soft-Margin SVM* by incorporating an upper bound to the number of training errors into Optimization Problem 1 and minimize it along with the weight vector:

Optimization Problem 2 (Soft-Margin SVM (PRIMAL))

$$minimize: \quad V(\vec{w}, b) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum_{i=1}^{n} \xi_i \tag{2.24}$$

$$subject\ to: \quad \forall_{i=1}^{n} : y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i \tag{2.25}$$

$$\forall_{i=1}^{n} : \xi_i > 0 \tag{2.26}$$

The ξ_i are called *slack variables*. To satisfy condition 2.25, those have to be greater than one if the corresponding training sample lies on the wrong side of the hyperplane. Therefore, $\sum_{i=1}^{n} \xi_i$ is an upper bound for the number of training errors. Parameter C can be used to control how errors are tolerated. Large values for C lead to the Soft-Margin SVM to behave similar to a Hard-Margin SVM because even slack variables with small values lead to large increases of the objective functions value. Small values for C will lessen the influence of the slack variables and hence allow for more errors. Finally, condition 2.26 prevents the assignment of zero to all slack variables, as this would be always optimal but does not take any of the ξ_i into account. Following the strategy to solve Optimization Problem 1 for Hard-Margin SVMs, Optimization Problem 2 is also transformed to its Wolfe dual due to numerical problems when solving it directly. See Joachims (2002) for further information.

Non-Linear SVMs

The SVMs mentioned so far can only handle linear classification problems. Even though text-classification problems are claimed to usually be linearly separable (Joachims 2002, Fan et al. 2008), some of them, along with many other real world problems, are not linearly separable. Fortunately, Boser et al. (1992) developed a method which enables the possibility to easily transform SVMs to non-linear learners. The attribute vectors $\vec{x}_i$ are basically just mapped into a higher dimensional space X' with a non-linear mapping function $\Phi(\vec{x}_i)$. The SVM then learns the linear maximum margin method as before but in the new feature space X' of higher dimension, where the data is now linearly separable. Even though the learned classification rule is linear in X', it is non-linear when transformed back to the initial feature space.

The following example, taken from Joachims (2002), illustrates the aforementioned transformation for two input variables x_1 and x_2. One chooses

$$\Phi((x_1, x_2)^T) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)^T \tag{2.27}$$

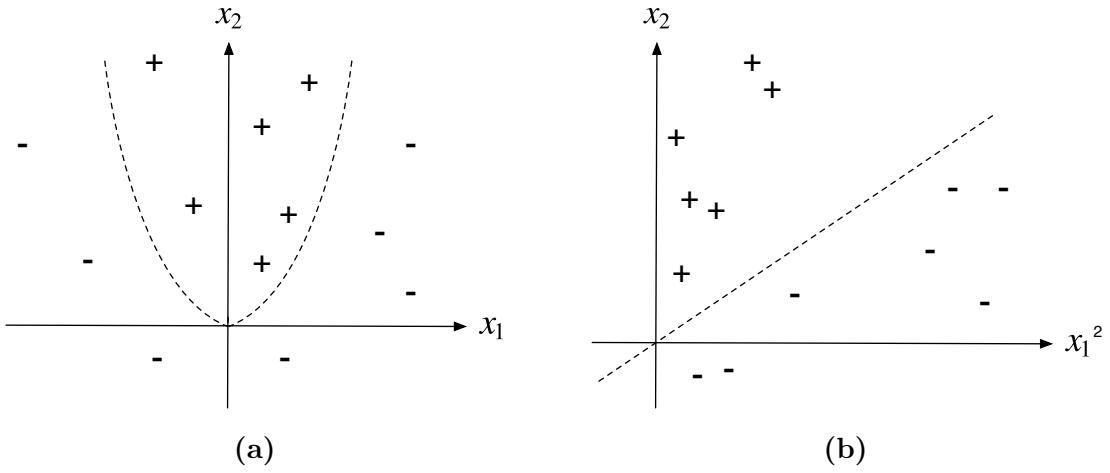


Figure 2.6: The training set shown in the left-hand graph (a) is obviously not linearly separable in (x_1, x_2). A non-linear transformation of the form (x_1^2, x_2) is depicted in the right-hand graph (b). Within this new space, the training examples are linearly separable. Also see Joachims (2002).

as a non-linear mapping function to transform the attribute vectors to X'. It is impossible to linearly separate the data as illustrated in the left-hand image (a) of figure 2.6. Yet, when mapping the data to another feature space using $\Phi(\vec{x})$, as shown in the right-hand side image (b) of 2.6, the data becomes linearly separable. One possible linear separator (although not with maximum margin) would be the weight vector $\vec{w} = (-1, 0, 0, 0, \sqrt{2}, 0)^T$ with $b = 0$ (it is illustrated as dotted line in both images of figure 2.6).

In general, the mapping function $\Phi(\vec{x})$ cannot be efficiently computed. Boser et al. (1992) have been able to solve this problem. They found it to be sufficient to compute the dot product $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ in the new feature space, when solving the dual optimization problems. For some special cases of $\Phi(\vec{x})$ those can be efficiently computed using so called *kernel functions* $\kappa(\vec{x}_1, \vec{x}_2)$. As long as those kernel functions satisfy Mercer's Theorem, they are guaranteed to compute the inner product of mapped vectors in the feature space X' (Vapnik 2000):

$$\kappa(\vec{x}_1, \vec{x}_2) = \Phi(\vec{x}_1) \cdot \Phi(\vec{x}_2) \ . \tag{2.28}$$

Depending on the choice of the kernel function, SVMs are able to learn polynomial classifiers, radial basis function (RBF) classifiers or two layer sigmoid neural networks:

$$\kappa_{\text{poly}}(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 \cdot \vec{x}_2 + 1)^d \tag{2.29}$$

$$\kappa_{\text{rbf}}(\vec{x}_1, \vec{x}_2) = exp(-\gamma(\vec{x}_1 - \vec{x}_2)^2) \tag{2.30}$$

$$\kappa_{\text{sigmoid}}(\vec{x}_1, \vec{x}_2) = tanh(s(\vec{x}_1 \cdot \vec{x}_2) + c) \ . \tag{2.31}$$

The kernel function for the mapping of the example from above is $\kappa_{\text{poly}} = (\vec{x}_1 \cdot \vec{x}_2 + 1)^2$. This is obviously much more efficient than enumerating all possible polynomial terms, like in polynomial regression.

The incorporation of the kernel functions into the learning process is done by replacing every occurrence of inner products within the dual optimization problems with the chosen kernel function. See Joachims (2002) for further details.

Multi-Class SVMs

The previous sections described SVMs build for two-class classification problems. Usually there are more than two classes needed in the field of Twitter sentiment analysis . Hence, the most widely used methods for solving problems with more than two classes using SVMs are presented here. The following explanations are based upon the descriptions in Hsu and Lin (2002).

Support Vector Machines for binary problems were already well developed when research for multi class problems began. Hence, the first and still most widely used ideas were based upon building multi class classifiers from multiple binary classifiers.

One of the first methods proposed and implemented was the *one-against-all* strategy (Bottou et al. 1994). Let k be the number of classes to be classified, then the method creates k SVM models. Each of the models - let m denote the index of one arbitrary model of the k models as an example - is trained with all examples of the mth class as positive examples and all other examples as negative examples. Given n training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where x_i are the feature vectors, $i = 1, \ldots, n$, and $y_i \in 1, \ldots, k$ is the class of the example x_i, the SVM with index m solves the problem:

$$minimize: \quad V(\vec{w}^m, b^m) = \frac{1}{2}\vec{w}^m \cdot \vec{w}^m + C\sum_{i=1}^{n} \xi_i^m \tag{2.32}$$

$$subject\ to: \quad \forall_{i=1}^{n} : y_i(\vec{w}^m \cdot \vec{x}_i + b^m) \geq 1 - \xi_i^m, \text{if } y_i = m \tag{2.33}$$

$$\forall_{i=1}^{n} : y_i(\vec{w}^m \cdot \vec{x}_i + b^m) \leq -1 + \xi_i^m, \text{if } y_i \neq m \tag{2.34}$$

$$\forall_{i=1}^{n} : \xi_i^m > 0 \tag{2.35}$$

Basically, this is the same optimization problem that was formulated for the soft-margin SVM (see section 2.2.2). The difference lies in the additional condition for treating all data examples from other classes than m as negative examples.

After solving the aforementioned optimization problem one has trained k SVM classifiers which provide k decision functions:

$$\begin{aligned} d^1(x) &= \vec{w}^1 \cdot \vec{x} + b^1, \\ &\vdots \\ d^k(x) &= \vec{w}^k \cdot \vec{x} + b^k, \end{aligned}$$

A classification function for the multi-class problem can then be defined as:

$$\text{classify}(x) := \underset{m=1,\cdots,k}{\arg\max}\, d^m \tag{2.36}$$

The classification result of the model with the highest margin - i.e. confidence - is chosen as final result for the multi-class classification.

The concept of *one-vs-one* Support Vector Machines was first used by Friedman (1996), Kreßel (1999). The basic idea is to construct $\frac{k(k-1)}{2}$ classifiers where each is trained using data from two of the k classes. Training each of the classifiers with data from the classes i and j can be formulated as the following optimization problem:

$$\textit{minimize}: \quad V(\vec{w}^{ij}, b^{ij}) = \frac{1}{2}\vec{w}^{ij} \cdot \vec{w}^{ij} + C\sum_{i=1}^{n} \xi_i^{ij} \tag{2.37}$$

$$\textit{subject to}: \quad \forall_{i=1}^{n} : y_i(\vec{w}^{ij} \cdot \vec{x}_i + b^{ij}) \geq 1 - \xi_i^{ij}, \text{if } y_i = i \tag{2.38}$$

$$\forall_{i=1}^{n} : y_i(\vec{w}^{ij} \cdot \vec{x}_i + b^{ij}) \leq -1 + \xi_i^{ij}, \text{if } y_i = j \tag{2.39}$$

$$\forall_{i=1}^{n} : \xi_i^{ij} > 0 \tag{2.40}$$

A commonly used classification function (according to Hsu and Lin (2002), as also suggested by Friedman (1996)) is the so called *"Max Wins"* strategy. For each of the k classes it is counted how many of the $\frac{k(k-1)}{2}$ have chosen the respective class. The class with the maximum number of votes is the class chosen as overall classification result. One drawback of the method is the possibility of a draw. If two classes have the same number of votes, Hsu and Lin (2002) suggest to chose the one with the smaller index, probably to enforce deterministic behaviour and keep the bias consistent.

2.2.3 Maximum Entropy Classifier

The main idea of *Maximum Entropy Classification* (MaxEnt) is to infer an unknown probability density function subject to a set of constraints without any *a priori* knowledge about the density function (Costache et al. 2006). This following short introduction to MaxEnt models is based closely upon the excellent remarks of Yu et al. (2011).

MaxEnt is derived from Logistic Regression which models conditional probability as:

$$\mathcal{P}_{\vec{w}}(y = \pm 1|\vec{x}) \equiv \frac{1}{1 + e^{-y\vec{w}\cdot\vec{x}}} \tag{2.41}$$

where $\vec{x}$ is the sample to be classified, y is the class label and $\vec{w} \in \mathbb{R}^n$ is the weight vector to be learned. Let the number of classes be $l = 2$, then for training data $\{\vec{x}_i, y_i\}_{i=1}^{l}$

with $\vec{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$ the following log-likelihood is minimized by logistic regression:

$$P^{LR}(\vec{w}) = C \sum_{i=1}^{l} \log(1 + e^{-y_i \vec{w} \cdot \vec{x}_i}) + \frac{1}{2} \vec{w} \cdot \vec{w} \tag{2.42}$$

with $C > 0$ being a penalty parameter. For more information on the optimization methods to train logistic regression, please see Yu et al. (2011).

MaxEnt is a multi-class generalization of Logistic Regression. The conditional probability in MaxEnt is modelled as:

$$\mathcal{P}_{\vec{w}}(y|\vec{x}) \equiv \frac{exp(\vec{w} \cdot \vec{f}(\vec{x}, y))}{\sum_{y'} exp(\vec{w} \cdot \vec{f}(x, y'))} \tag{2.43}$$

where $\vec{x}$ represents a context, y is the class label of $\vec{x}$ and $\vec{w}$ is the weight vector to be learned. The features which are extracted from a context and its label are determined by the function vector $\vec{f}(x, y) \in \mathbb{R}^n$. Given N training samples $\{(x, y)\}$ which are grouped into l (number of classes) unique contexts $\{x_i\}$, the empirical probability distribution can be calculated as $\widetilde{\mathcal{P}}(x_i, y) = \frac{N_{x_i,y}}{N}$ where $N_{x_i,y}$ is the count of occurrences of (x_i, y) in the training data. Training MaxEnt is then done by minimizing the following regularized negative log-likelihood:

$$\arg\min_{\vec{w}} P^{ME} = \sum_{i=1}^{l} \widetilde{\mathcal{P}}(x_i) \log \left(\sum_{y} exp(\vec{w} \cdot \vec{f}(x_i, y)) \right) - \vec{w} \cdot \widetilde{\vec{f}} + \frac{1}{2\sigma^2} + \vec{w} \cdot \vec{w} \tag{2.44}$$

with $\widetilde{\mathcal{P}}(x_i) = \sum_y \widetilde{\mathcal{P}}(x_i, y)$ being the marginal probability of x_i, σ being a penalty parameter and

$$\widetilde{\vec{f}} = \sum_{i=1}^{l} \sum_{y} \widetilde{\mathcal{P}}(x_i, y) \vec{f}(x_i, y) \tag{2.45}$$

being the expected vector of $\vec{f}(x_i, y)$. For further information on how to solve the optimization problem please see Yu et al. (2011) and the multitude of references therein.

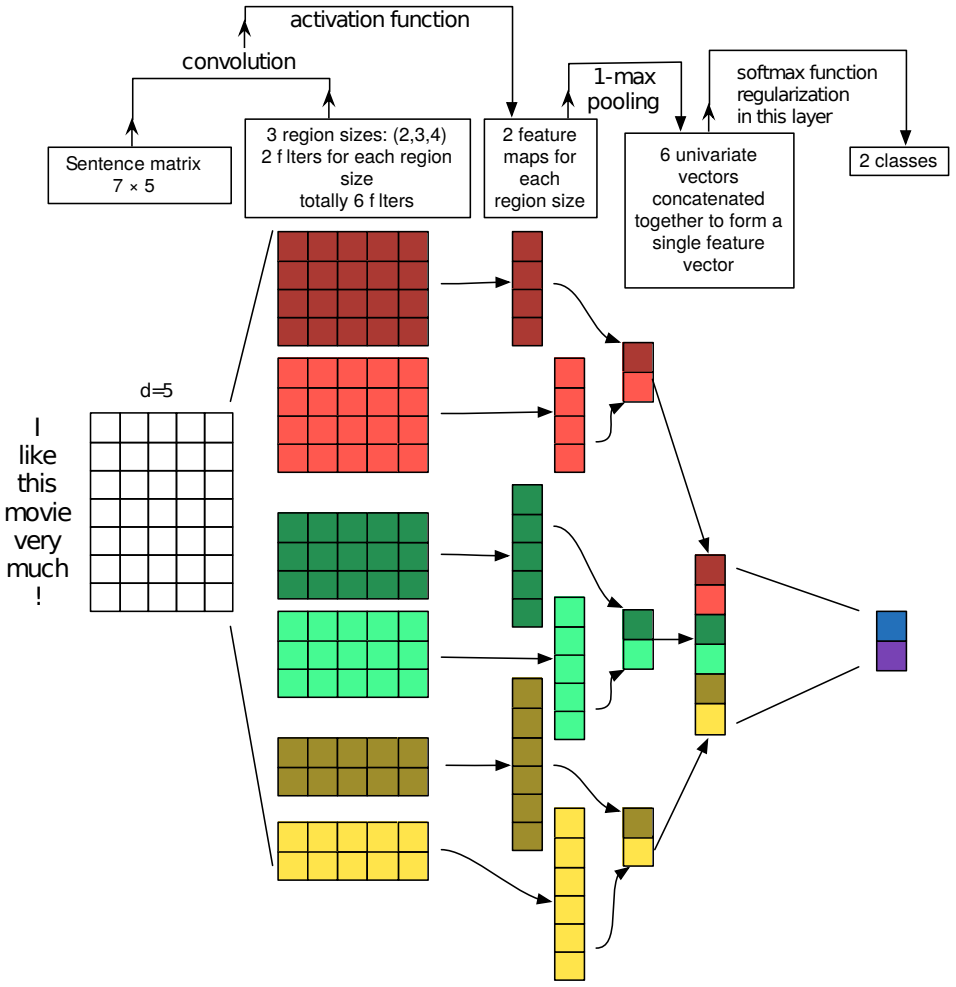


Figure 2.7: Schematic illustration of a deep convolutional neural network for text data classification, taken from Zhang and Wallace (2015)

2.2.4 Deep Convolutional Neural Networks

Artifical neural networks were originally inspired by the human brain. They consist of a directed graph of nodes called *neurons* which exist in multiple fully interconnected layers. The connections have certain weights and the neurons posses a non-linear activation function. Artificial neural networks are often trained in a supervised manner using techniques based upon *backpropagation*. For a more thorough introduction to the topic see for example Haykin (1994).

Recent advances in technology made it possible to train networks with increasing numbers of large hidden layers and the term *deep neural network* was coined. In combination with *convolution layers* and word embeddings[17] those networks can be successfully harnessed for the analysis of text data[18]. For a more detailed introduction see Kim (2014). In the following, the basic idea of the aforementioned methods is outlined.

One of the main benefits of using a *deep convolutional neural network* (DCNN) is that

[17] See section 2.1.2.
[18] See for example Collobert et al. (2011), Kim (2014), Zhang and Wallace (2015).

no manual feature engineering and just very little preprocessing is necessary. The input of those networks is a one-hot vector indicating the presence or absence of words from the vocabulary in the text as would be the case for bag-of-word models[19]. Features are learned by the network as part of the training process. Each text (e.g. "I like this movie very much!") can now easily be transformed to a sequence of indices which represent the present words, e.g. [213, 4123, 12345, 422373][20].

The first layer in a DCNN is often the so called *embedding layer* which consists of a list of d dimensional word embeddings. Using those embeddings the network concatenates the embeddings for the n indices from the given input and creates a sentence matrix (first column in figure 2.7) which now represents the input sentence as a list of d dimensional word embeddings $v = (v_1, \ldots, v_n)$ with $v_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$. The embedding layer can be initialized randomly and its weights can then be trained with the network, but usually it is initialized with previously acquired word embeddings that have been generated in an unsupervised manner as described in section 2.1.2.

Behind the embedding layer one usually finds a *convolution* layer which is the main characteristic of the network. The convolution layer's output is a set of feature maps of various region sizes (the example in figure 2.7 consists of 6 feature maps, illustrated in the third column). A feature map of region size r consists of $n - r + 1$ components where the k components F_k of the map are computed for all $1 \leq k \leq n - r + 1$ as:

$$F_k = g\left(b + \sum_{i=1}^{r} v_{k+i-1} \cdot M_k\right)$$

where M_k represents the kth row of a $(d \times r)$-Matrix M which is the *filter* that is used for generating the feature map (shown in column two of figure 2.7). The symbol g stands for the activation function and b is the bias vector as typically used in neural networks. Basically one can picture that process as the filtering matrix sliding over the sentence matrix row by row and at each step a component of the feature map vector is generated by applying the activation function, weights and biases. Note that at this point the size of the feature map depends on the input length n. The reasoning behind such feature maps and filters is inspired by the animal visual cortex and its receptive field. Each filter is hopefully able to learn to recognize certain more abstract structures which can be detected in the respective sliding window. This approach seems to be not as complex as trying to grasp the full input matrix as a whole. The activation function used between the convolution layer and the following pooling layer is usually one of:

linear activation (linear) denotes the identity function $g = \mathrm{id}$

tangens hyperbolicus (tanh) $g(x) = \tanh(x)$:

[19] Compare 2.1.1.

[20] Note that the sequence is shorter than the initial sentence. Especially when dealing with informal language such as that in tweets, not all words from the initial text are available in the underlying vocabulary.

This function is often used in neural networks due to its similarity to the sigmoid function $S(x) = (1+e^{-x})^{-1}$ (it is $\tanh(x) = 2S(2x)-1$) but offers some numerical advatages[21].

rectified linear unit (ReLU) $g(x) = \max(0, x)$:

This activation function offers a more realistic activation than other commonly used activation functions which are based on the Heaviside function (Glorot et al. 2011) and has successfully been used especially in deep learning architectures (LeCun et al. 2015).

The feature maps generated by the convolution layer have a variable size which depends on the input length n. To reduce the intermediate data to a fixed length vector, a *pooling layer* is used (column four in figure 2.7). The most commonly used pooling function is *1-max pooling* which simply chooses the highest value from each feature map and concatenates the chosen values to a single feature vector. Intuitively, this approach also makes sense: When imagining each filter as a detector for a certain abstract structure it seems sufficient to only take its highest activation into account. Moreover, this is the place where regularization is usually integrated. A common approach to prevent overfitting in deep neural networks is *dropout*. Basically, each neuron in the feature vector is set to zero with a certain probability in each training step to ensure proper generalization.[22]

Finally, a fully connected layer[23] with as many output neurons as there are classes in the classification problem (two in the example in figure 2.7) makes up the final part of the network. As activation function for the output neurons one uses the softmax function $\sigma\colon \mathbb{R}^{|C|} \to \mathbb{R}^{|C|}$[24] to estimate the probabilities of the input belonging to each of the possible classes.

[21] See for example LeCun et al. (2012), section 4.5.

[22] See Srivastava et al. (2014) for further details.

[23] Note that one can use multiple such layers as is often done in image processing tasks, however, for text data one layer is sufficient in most cases. See Kim (2014) for further evidence.

[24] See equation 2.1 in section 2.1.2.

3 History and State-of-the-Art

Twitter sentiment analysis is a relatively young field of research, that gained increasing attention since 2009. First, the history and the early approaches are reviewed. Next, the development of the various proposed methods is discussed in chronological order. The aforementioned sections are based upon the proceedings of the International Workshop on Semantic Evaluation[1] which provides a representative snapshot of the field. Finally, the development of the field is summarized to provide an overview.

3.1 Emergence of the Field

This subsection is inspired by previous work of the author (Haldenwang 2013) in collaboration with Neubauer (2014). An overview of the field from its emergence in 2009 till 2012 is provided here. For a more detailed review of this period see Neubauer (2014).

One of the earliest works in the field is Go et al. (2009). In this work the authors transfer known methods - mainly supervised machine learning - for sentiment classification of blog articles or product reviews to Twitter. They attempted to solve the two class classification problem, distinguishing between positive and negative sentiment. Acquiring labeled Twitter data for training supervised machine learning methods is not an easy task. While product reviews include a rating of the reviewer which correlates to the reviews sentiment, tweets do not necessarily include such inherent labels. Hence, the usage of emoticons as noisy labels, introduced by Read (2005), was transferred the to the domain of Twitter sentiment analysis to acquire training data. The underlying hypothesis is that tweets containing positive emoticons such as "`:-)`" are of positive sentiment and tweets including negative emoticons such as "`:-(`" are of negative sentiment. While this hypothesis obviously does not always hold up, the resulting noise can be neglected when using a sufficient amount of data. Overall, 1.6 million tweets have been collected as training data, and were labeled with a sentiment using the emoticons. Moreover, the authors used various preprocessing techniques to reduce the dimension of the data. They attempted to correct spelling mistakes and replaced Twitter specific entities such as user names with wildcards, for example "@JohnDoe" becomes "USERNAME" and URLs such as `http://www.example.com` become "URL". The applied preprocessing techniques reduced the number of distinct tokens to about 48.6% of their original size. Feature extraction methods considered by the authors were unigrams, unigrams with part-of-speech (POS) tags and bigrams. These feature extraction methods were evaluated for Support Vector Machines (SVMs)[2], Naive Bayes

[1] See Nakov et al. (2013), Rosenthal et al. (2014; 2015), Nakov et al. (2016).
[2] See section 2.2.2.

Classifiers (NBCs)[3] and Maximum Entropy Classifiers (MaxEnt)[4]. To evaluate the performance the authors manually annotated 182 positive and 177 negative tweets. SVMs performed best using unigrams with an accuracy of 82.2%. Using Naive Bayes classifiers with La Place smoothing and a combination of unigram and bigram features achieved an accuracy of 82.7%. The lead was taken by the Maximum Entropy classifier, reaching 83% accuracy, also using the combination of unigrams and bigrams. While the results should be considered with caution due to the small size of the test set, they provided inspiration and direction for the field with the main contribution being the usage of the emoticons as noisy labels which is still relevant nowadays.

Pak and Paroubek (2010) followed the approach of Go et al. (2009), they used emoticons as noisy labels for training data. In addition, they considered the class *neutral*. Neutral training data was obtained by gathering tweets from newspapers. Regarding preprocessing, tweet specific entities along with emoticons have been removed from the training data. Furthermore, stopwords are also excluded. Basis of the method is a Naive Bayes classifier with a combination of bigrams and POS tags as features. Their goals were not to find an algorithm outperforming others but to investigate the usability of features. They found that bigrams perform better than unigrams and trigrams and that large numbers of training samples increase the classification accuracy. In the end, they introduced two measures to determine the quality of features, which did not lead to an improvement in general.

Agarwal et al. (2011) introduced a special set of features for SVMs, called *senti features*. Those senti features can be natural numbers, like the count of negations within the tweet, real numbers, such as the percentage of capitalized text, or binary, for example the presence of an exclamation mark. Incorporating their senti features along with the unigram features raised the accuracy by about 4%. Nevertheless, the overall accuracy is just 75.39%. As test set they acquired 11,875 manually annotated tweets from a commercial source. Those tweets were collected by sampling the Twitter API, translating them using Google Translate[5] and finally letting people label them. After the removal of tweets which have been labeled as *junk*, 8,752 tweets were left to work with. However, the translation step is fairly questionable. Even though Google Translate works remarkably well, a translated tweet will differ a lot from the same tweet originally written in English. Google Translate would, for example, probably not output the informal language most authors of tweets use. Still, the results suggest that handcrafted features for SVMs can possibly yield improvements.

Liu et al. (2012) introduced the idea of using a combination of noisy labels like emoticons and manually annotated data. A Naive Bayes classifier is the basis for their method. Using Jelinek-Mercer smoothing, they combine a manually annotated training set and an emoticon model with the features being unigrams only. In addition, tweet specific entities were replaced with wildcards, stopwords were removed and stemming was performed. Both classification problems, the subjectivity classification (distinguishing between neutral tweets and those carrying sentiment) and the two class classification of positive and negative sentiment (polarity classification) were investigated. While the

[3] See section 2.2.1.
[4] See section 2.2.3.
[5] `http://translate.google.com`, last checked 21.07.2017.

distant supervised language model (emoticons) achieved an accuracy of only 72%, the incorporation of just 768 manually annotated tweets raised this up to about 82% for polarity classification. The results for subjectivity classification are very similar. As a dataset they used the Sanders corpus[6], consisting of 5,513 manually annotated tweets. After filtering out non-English and spam tweets, 3,727 tweets were left for experimenting. Furthermore, those tweets were collected querying the Twitter API for only four keywords (Apple, Google, Microsoft and Twitter), which makes the corpus rather biased towards tweets about those entities. Nevertheless, it serves the purpose of validating the ideas of Liu et al. (2012). But for a general comparison to other methods it is not suited well.

Saif et al. (2012a) use a Naive Bayes classifier where they combined different feature types with smoothing. Their basic model consists of unigram features, smoothed with semantic features. They found what they call *sentiment-topic features* to perform best. Sentiment-topic features are generated by a clustering method for words, which clusters them by sentiment and topic. The unigram model is then augmented with the sentiment topics present in the tweets. For preprocessing, they used tweet specific entity replacement and repeated letter spelling correction. To compare their approach with others, the dataset of Go et al. (2009) has been used. Although the approach outperforms the other approaches by a few percent (it achieves 86.3% accuracy), the test suffers from the aforementioned drawbacks. Saif et al. (2012a) have done some pioneer work regarding the integration of semantics into Twitter sentiment analysis.

Bakliwal et al. (2012) use an unigram model as a baseline. Afterwards, preprocessing techniques are incorporated step by step to monitor their effectiveness. Their results indicate that spelling correction, stemming and stop word removal increase the accuracy. The best results were achieved by SVMs in conjunction with sentiment feature vectors, similar to those of Agarwal et al. (2011). On the dataset from Go et al. (2009) the method reached 88% accuracy. Furthermore, another dataset called *Mejaj* (Bora 2012) is used. It is very similar to the other one, but instead of using emoticons as noisy labels, it uses a handcrafted list of 40 words, 20 for each sentiment. Those test sets are neither very large, nor is there any information about the manual annotation process and the quality criteria.

Most of the early approaches suffer from a lack of comparability as it is almost always the case in a new field. Not many datasets were publicly available. Those which have been shared only included IDs of tweets, since the Twitter usage terms prohibit the transfer of the actual text data. When another researcher wanted to use the same dataset one had to download the tweets again. Unfortunately, some of the tweets were no longer available. Even if the same dataset was used the results may have differed because of the missing data. However, there were multiple datasets which have been used for evaluation, resulting in multiple contradicting claims and result. The datasets that were used often were of small size and no information about their quality - such as inner annotator agreement - were given.

Driven by the desire to get a deeper insight into what combinations of preprocessing, feature extraction and machine learning algorithms actually yield the best results, a

[6] `http://www.sananalytics.com/lab/twitter-sentiment/`, last checked on 14.07.2017.

comparative study was conducted on a newly created dataset of high quality (Neubauer 2014). From June 2012 to August 2013 a corpus of 43 million tweets was collected. Additionally, more than 10 million tweets including emoticons were collected in that time frame. The large time frame reduces the topical bias of the dataset. Most of the other datasets of the time were collected in a span of a few weeks, and hence they are biased towards the hot topics of the time. To acquire high quality test data, tweets were randomly sampled from the 43 million collected tweets. Each tweet was annotated by two humans with a sentiment of either *positive*, *negative* or *neutral* while also providing the option *garbage* if the labeler did not understand the message or could not assign one of the other labels. Tweets labeled *garbage* once, were instantly discarded. The test set is then constructed by picking only the tweets with two identical labels, resulting in 910 positive, 1,328 negative and 966 neutral tweets. For further statistics of the dataset see Neubauer (2014). On this dataset various combinations of classification algorithms, feature types and preprocessing techniques have been evaluated. The results showed that SVMs consistently seem to outperform NBCs. N-Grams with $n = 2$ outperformed other n-grams and subgrams. Negation annotation was the only preprocessing method which actually yielded improved classification accuracy, the other evaluated methods sometimes even resulted in decreases of performance. Moreover, Neubauer evaluated the integration of semantic knowledge acquired by using semantic similarity measures such as *Pointwise Mutual Information* (Turney 2001) and the *Bidirectional Co-Occurrence Measure* (Neubauer et al. 2013) incorporated into the classification process as suggested by Turney and Littman (2003) and Saif et al. (2012a). Unfortunately, the performance decreased for both methods and similarity measures.

To summarize the early stages of the field, one could say that most approaches consisted of training machine learning algorithms - mainly SVM and NBC - using emoticons as noisy labels. Most of the evaluations had been done on different datasets - of often small size and unknown quality -, which made comparing results almost impossible. A larger comparative study was conducted, which indicated that many of the proposed methods yield no significant improvements in classification performance or even resulted in decreased performance.

3.2 SemEval 2013

In 2013 the organizers of the *International Workshop on Semantic Evaluation* also were of the opinion that "[...] research has been hindered by the lack of suitable datasets, complicating the comparison between approaches [...]" and created a task dedicated specifically to Twitter sentiment analysis (Nakov et al. 2013). Since its introduction the SemEval Twitter sentiment analysis task has become a suitable reflection of the state-of-the-art of the field. The task consisted of two subtasks, *Subtask A: Contextual Polarity Disambiguation* (expression-level) and *Subtask B: Message Polarity Classification* (message-level). Since this thesis' main focus is message-level polarity classification, subtask A is discussed no further here. To also demonstrate that sentiment analysis systems designed for tweets can be transferred to other domains, a SMS dataset is provided for each subtask along with the tweet dataset.

For the creation of the tweet dataset, millions of tweets were collected in the period from January 2012 to January 2013. From those tweets named entities were extracted and popular topics were identified using the extracted entities. Testing tweets were chosen from separate time periods and topics than the training data. Finally, a filter was applied discarding tweets which contained no sentiment bearing words, determined by SentiWordNet (Baccianella et al. 2010). The authors themselves note the bias towards popular topics and towards terms occurring in SentiWordNet. Sentiment labels for the messages were obtained using *Amazon Mechanical Turk*. To identify a tweet's sentiment a simple majority vote was conducted on the labels assigned by the turkers.

Competing systems had to classify the given messages as either *positive*, *negative* or *objective*. Systems could be *constrained* or *unconstrained*, where constrained systems only used the provided training data while unconstrained systems were allowed to use additional resources for training. Performance was measured using the macro-averaged F-score of the classes *positive/negative*. Contestants taking part in subtask B consisted of one semi-supervised system, the rest was supervised. Most of the systems consisted of SVMs (8), NBCs (7) or MaxEnt classifiers (3). There were also ensembles of classifiers, one linear classifier and manually created rule sets. In total there were 36 constrained and 15 unconstrained systems. The following paragraphs present a summary and discussion of the top systems.

3.2.1 NRC-Canada: Building the State-of-The-Art with Sentiment Lexicons

Mohammad et al. (2013) as team NRC-Canda contributed the top performing system to the competition. They combined various feature types and preprocessing methods and trained SVM models with them, building upon the early work in the field. For training they used only the human annotated tweets provided for the task.

Tweets were preprocessed by normalizing URLs to "`http://someurl`" and user names to "@username". Next, the tweets' content was tokenized and POS tagged. Standard features such as n-grams, non-contiguous n-grams and subgrams were used. Additionally, Mohammad et al. (2013) introduced a number of hand-crafted features. Words with all characters in upper case were counted, the number of hashtags was noted, contiguous sequences of exclamation marks, question marks or combinations of both were accounted for. They also counted the number of elongated words such as "looooove". Negated contexts were explicitly handled by annotating the negated tokens with the postfix *_NEG*, following the procedure of Pang et al. (2002). Moreover, the presence or absence of tokens from 1,000 token clusters as alternative representation of the content provided by the CMU pos-tagging tool (Gimpel et al. 2011) were incorporated. Presence or absence of positive and negative emoticons was incorporated as binary feature for a given list of emoticons.

The most notable innovation of the approach is the ponderous usage of various sentiment lexicons. Three existing manually created lexicons were used: The NRC Emotion Lexicon (Mohammad and Turney 2010), the MPQA Lexicon (Wilson et al. 2005) and

the Bing Liu Lexicon (Hu and Liu 2004). Additionally the authors created two automatically generated sentiment lexicons. From April to December 2012 they gathered 775,000 tweets by querying the twitter API for tweets including hashtags from a collection of 78 seeds words such as *#good*, *#bad*, *#terrible* and *#excellent*. The tweets were labeled with *positive/negative* according to the sentiment of the hashtag present. For each token w a score was computed as:

$$score(w) = PMI(w, \text{positive}) - PMI(w, \text{negative}) \tag{3.1}$$

where PMI is the pointwise mutual information. A score above zero indicates positive sentiment, a score below zero indicates negative sentiment. Scores were not only generated for unigrams but also for bigrams and non-contiguous combinations of unigrams and bigrams. The second lexicon was generated by applying the same procedure to the sentiment140 corpus from Go et al. (2009) where the tweets are labeled *positive/negative* by the presence of emoticons instead of using hashtags like before. For each of the five lexicons separate feature sets were generated for unigrams, bigrams and non-contiguous bigrams. The feature sets were generated as was described in section 2.1.1.

Using all the aforementioned features in combination the approach achieved an F-score of 69.02 on the SemEval evaluation dataset and ranked first of all submitted systems.

To assess the importance of the various feature groups, the authors performed an ablation experiment where they repeatedly performed the classification but left out one group of features at a time. The results are shown in table 3.1.

Experiment	**F-Score**
all features	69.02
- all lexicons	60.42 (-8.60)
- all manually created lexicons	67.45 (-1.57)
- all automatically generated lexicons	63.78 (-5.24)
- Senti140 lexicon	65.25 (-3.77)
- Hashtag lexicon	65.22 (-3.80)
- n-grams and subgrams	61.77 (-7.25)
- only n-grams	64.64 (-4.38)
- only subgrams	67.10 (-1.92)
- explicit negation handling	67.20 (-1.82)
- POS tags	68.38 (-0.64)
- token clusters	69.01 (-0.01)
- encoding based features (elongated words, emoticons, punctuations, all-caps count, hashtags)	69.16 (+0.14)

Table 3.1: The table shows the macro-averaged F-scores obtained on the SemEval test set with one feature group removed per experiment. The numbers in brackets indicate the difference to using all features. This data was taken from Mohammad et al. (2013).

It can be noted that the biggest impact on the performance was made by the lexicon based features with the automatically generated lexicons having a much higher impact than the manually created lexicons. Both the Senti140 lexicon and the Hashtag lexicon yielded very similar benefits, the impact of each still being higher than all manually created lexicons combined. N-gram features had an impact of similar magnitude with word level n-grams being of higher importance than subgrams. Negation handling and POS tags are of notable but not very high impact. Token clusters only differ by -0.01% which is probably not significant. Surprisingly, the encoding based features yielded decreases in performance. While the decrease is of small magnitude, it can be stated that they obviously did not yield any benefit and hence should be discarded for future work.

All in all, the NRC-Canada used well-known and established approaches from the early stages of the field and successfully enriched them with lexicon based features and explicit negation handling.

3.2.2 GU-MLT-LT: Linguistic Features and Stochastic Gradient Descent

Team GU-MLT-LT ranked second with their contribution "Sentiment Analysis of Short Messages using Linguistic Features and Stochastic Gradient Descent" (Günther and Furrer 2013). Their model is built upon twitter specific preprocessing and linguistic features such as word stems, negation handling and word clusters.

Tokens were stored in three versions from which the features are computed later. Version one was the *raw* version of the token as it appeared in the message. Version two was a *normalized* version where all characters where changed to lower case and digits were replaced by 0. The *collapsed* version trims elongated words such as "looooove" to the version of the word with only two repetitive letters: "loove".

The feature set used consisted of unigrams represented by the normalized tokens, stems of the collapsed tokens, presence of words clusters as provided by Owoputi et al. (2013) and accumulated and summed SentiWordNet scores (Baccianella et al. 2010) for the positive and negative class.

Classification was done by training three linear models with stochastic gradient descent in an one-vs.-all manner and choosing the label with the highest score. The loss function used was *hinge loss* and the authors tried to prevent overfitting using *elastic net regularization*.

To assess the impact of the chosen feature groups the authors performed an ablation experiment leaving out one feature group at a time and measuring the impact on the systems performance. Performance was measured by the average F-Score of the positive and negative classes as described in task description of SemEval Task 2. Table 3.2 shows their results. Unigram features had the biggest impact on performance while all the other features also yielded small benefits. The authors mentioned that they tried to

Feature Set	F-Score
ALL	65.54
-Stemming	-0.385
-Word Clusters	-0.835
-SentiWordNet Scores	-0.23
-Negation Annotation	-0.30
-Unigrams	-3.83

Table 3.2: Results of the ablation experiment performed by Günther and Furrer (2013).

incorporate tweet specific part-of-speech tags but discarded those features due to no noticeable benefit.

In summary, one can say that simple linear models seem to perform on par with other classification algorithms. The authors found that stemming, word clusters, SentiWordNet and negation annotation have positive impact on classification performance.

3.2.3 teragram: Rule-Based Detection of Sentiment Phrases

Team teragram managed to rank third with their approach "Rule-based detection of sentiment phrases using SAS Sentiment Analysis" (Reckman et al. 2013). According to the authors their system is "entirely rule-based, and the rules are hand-written" and it can be seen as "a highly phrasal sentiment lexicon". The system is based on a domain independent sentiment taxonomy which was slightly modified for the purpose of Twitter sentiment analysis.

Their rules are hand defined patterns which match sequences of words. Rules are lists of concepts which can be used in other rules. An example rule provided in the paper is the following:

```
_def{Negation} _def{PositiveAdjectives}
```

Phrases starting with a negation (which is determined by a predefined list of negations) and followed by an adjective expressing positive sentiment (also determined by a predefined list) are matches by this rule. Taking new structures encountered in the training data provided by SemEval the authors added some special rules to the general purpose taxonomy. In total, the system contains of 10,000 words and 7,000 phrases. The authors note that an expert would take about 8 weeks to create a system for another language. Their adaptations for the contest took two man-weeks. Classification was done by summing up the number of weighted matches for each class and choosing the class with the highest score.

For the message level task the system achieved an F-Score of 0.6486[7].

[7]This is the result taken from Nakov et al. (2013) since Reckman et al. (2013) computed the average F-score of all three classes, not just positive and negative as it should have been done for the contest.

All in all, the authors build a very large rule-based system by manually creating rules to match phrases. The task of creating the rules is very labor intensive and cannot easily be reproduced. However, the approach performed on a competitive level with the machine learning based approaches.

3.2.4 BOUNCE: Rich Feature Sets

Team BOUNCE ranked fourth with their approach "BOUNCE: Sentiment Classification in Twitter using Rich Feature Sets" (Kökciyan et al. 2013). They trained multiple classifiers in an one-vs-all manner using manually engineered rich feature sets.

Preprocessing of the tweets consisted of tokenization and POS tagging using the Ark Tokenizer from CMU (Gimpel et al. 2011).

Various lexicons have been used by the authors. They harnessed AFINN (Nielsen 2011) and SentiWordNet (Baccianella et al. 2010) as sentiment lexicons, Urban Dictionary[8] for slang handling and a list of sentiment words enhanced with sentiment bearing words from the SemEval training dataset which they called *oursent*.

Using surface structure features of the tweets and the aforementioned lexicons the authors created a large number of features to be used for classification:

twitter-tags: Presence or absence of emoticons, user mentions hashtags and URLs
repetition: Elongated words such as 'nooooooooooooo'
lastword: The last word in the tweet and its shape, for example the shape of 'OmG' would be 'XxX'
chat: A manually created list of chat abbreviations was used to expand the abbreviation to the initial words
interjection: Interjections such as 'hurraaah' or 'looool'
negation: The count of negation words
hash: Hashtag tokens, and polarity according to oursent word list
oursent: Each word that is present in the word list and also the sequence of polarity if multiple words are present
phrases: n-grams of size $n > 1$ which exist in the sentiment words list
afinn-phrases: Sentiment of phrases as provided by the AFINN lexicon. Phrases are removed and the remaining words are handled in isolation to create additional features
emo: A manually created list of emoticons is used to transform the emoticons to tokens such as 'HAPPY' which are then used as features

For a more detailed description of how exactly the features are computed and for further examples see Kökciyan et al. (2013).

For classification the authors trained MaxEnt and Naive Bayes Classifiers in a one-vs-all manner for each class. They chose the best performing classifier per class. To classify a tweet the result with the highest probability is chosen. For the message level tweet task they used MaxEnt for the positive class and Naive Bayes for the negative class.

[8] See `http://www.urbandictionary.com`, last checked on 14.07.2017.

Table 3.3 shows the impact of the various feature groups. The authors started with only the *oursent* features as baseline and subsequently added more feature sets to assess their impact.

Features	**Averaged F-Score**
oursent (baseline)	58.69
+ afinn-phrases	64.64
+ tags + hash	65.43
+ interjection + chat	65.53
+ emo + lingemotion	65.92
+ repetition + lastword	66.01
+ negation + others	66.32

Table 3.3: Macro-averaged F-score on Task B development set, taken from Kökciyan et al. (2013)

The feature set with the highest impact is *afinn-phrases* which yields a performance increase of about 6% on the development set. The remaining feature sets yield only very small benefits. However, none of them lead to a decrease in performance.

Notable about this approach is the absence of unigram features which are present in almost every other approach. Moreover, the authors introduced many new features which have not been used anywhere else before and showed their viability.

3.2.5 KLUE: Simple and Robust Methods

Team KLUE managed to rank at position five with their approach "Simple and robust methods for polarity classification" (Proisl et al. 2013). The authors harnessed n-gram features, a sentiment lexicon and a manually created list of emoticons and internet slang abbreviations as features for various supervised machine learning algorithms for classification.

Bag-of-word feature types that were used consisted of unigrams, a combination of unigrams and bigrams and unigrams with negation annotation. Preprocessing was done by performing downcasing and white space tokenization. Moreover, URLs and user IDs were replaced with placeholders. Punctuation was completely removed. Finally, all tokens were stemmed and the total number of tokens was included as feature.

Additionally, a feature set based on a manually created sentiment lexicon (Nielsen 2011) was created. The lexicon consists of 2,476 words and a sentiment score $s \in [-5; 5]$ where $s < 0$ indicates negative sentiment. Team KLUE felt that the lexicon needs to be extended and so they added about 400 distributionally similar words based on their co-occurrence with words from the lexicon in a large text corpus. Using the created lexicon they computed the following features per tweet:

- number of positive tokens
- number of negative tokens
- number of tokens expressing a sentiment with regard to the lexicon
- arithmetic mean of sentiment scores from tokens known to the lexicon

The last feature set used consists of 307 emotion markers such as emoticons and internet slang abbreviation tokens which the authors manually labeled as negative (-1), neutral (0) and positive (1). Creation of the respective feature set is similar to the lexicon features:

- number of positive markers
- number of negative markers
- number of known markers
- arithmetic mean of all the markers' labels

Evaluation was done using the test set provided by SemEval. The authors compared the performance of Support Vector Machines, Naive Bayes Classifiers and Maximum Entropy Classifiers. The results are summarized in table 3.4.

Classifier	**BOW**	**Lexi.**	**Lexi. Ext.**	**Markers**	**F-Score**
NBC	uni	-	-	-	0.5724
SVM	uni	-	-	-	0.5648
MaxEnt	uni	-	-	-	0.5907
MaxEnt	uni+bi	-	-	-	0.6019
MaxEnt	uni_{neg}	-	-	-	0.6041
MaxEnt	uni_{neg}	+	-	-	0.6261
MaxEnt	uni_{neg}	-	-	+	0.6100
MaxEnt	uni_{neg}	+	+	-	0.6185
MaxEnt	uni_{neg}	+	+	+	0.6306
MaxEnt	uni_{neg}	+	-	+	0.6353
MaxEnt	uni + bi	+	-	+	0.6370
MaxEnt	uni + bi	+	+	+	**0.6386**
MaxEnt	-	+	+	+	0.5226

Table 3.4: Results shown are taken from Proisl et al. (2013). The column *BOW* indicates the bag-of-word model used. The columns *Markers*, *Lexicon* and *Lexicon Extended* indicate the presence (+) or absence (-) of the respective feature sets described above.

Noting the MaxEnt classifier outperforms SVM and NBC using unigrams by a relative large margin the authors decided to only investigate the MaxEnt closer. Combining unigrams and bigrams yields notable improvements over using unigrams in isolation.

Unigrams with negation annotation perform roughly on par with unigrams+bigrams. Incorporation of the unextended sentiment lexicon yielded a big improvement. Adding the extension to the lexicon performed by the authors led to a small improvement for unigrams+bigrams but led to a decrease of performance for unigrams with negation annotation. Leaving out bag-of-word features led to a significant drop in performance.

The authors found that MaxEnt outperforms NBC and SVM, demonstrated the viability of sentiment lexicons and negation annotation and showed that combining different n-gram feature types can also yield improvements.

3.3 SemEval 2014

"SemEval-2013 Task 2: Sentiment Analysis in Twitter" (Nakov et al. 2013) was continued as "SemEval-2014 Task 9: Sentiment Analysis in Twitter" (Rosenthal et al. 2014). The subtasks A and B were the same as in 2013, contextual polarity disambiguation and message level polarity classification. Still, the authors extended the test datasets with additional tweets and off-domain sentences. The first addition consisted of about 1,800 tweets collected and annotated with the procedure used in 2013. Moreover, the authors felt the need to test the handling of sarcasm, making use of tweets containing the "#sarcasm" hashtag. Sarcastic tweets should not be taken into account literally (González-Ibánez et al. 2011, Liebrecht et al. 2013), but their sentiment should be flipped to correctly account for the opinion of the user. For the 2014 task, about 85 sarcastic tweets were added to the test set. Finally, about 1,100 sentences from LiveJournal were added to the test set to assess the performance of the twitter specific systems on other domains. For the evaluation the 2013 test set and the new 2014 test sets were mixed and shuffled to ascertain the participants could not guess from which set a message was coming and had to make predictions on all test sets. Furthermore, the participants did not know in advance where the additional test sets originated.

Scoring of the submissions was done as it was in 2013. However, there was only one ranking for both constrained and unconstrained systems in 2014.

All in all, trends were very similar to 2013, probably because many of the teams took part repeatedly. The most popular classifiers were still Naive Bayes, SVM and MaxEnt. Additionally, two deep learning based systems were submitted.

The following sections provide a more detailed summary of the top performing systems of 2014.

3.3.1 TeamX: Enhanced Lexicon Mapping and Weighting Scheme for Unbalanced Data

The first ranking system in 2014 was "TeamX: A Sentiment Analyzer with Enhanced Lexicon Mapping and Weighting Scheme for Unbalanced Data" (Miura et al. 2014). In general, their approach is similar to those of the systems performing well in 2013. They also performed specific normalization for the different lexical resources and introduced a

weighting scheme to bias the classifier's output according to the weight of the polarities in twitter data.

Preprocessing steps used were unicode normalization, downcasing and the replacement of URLs with the token "URL". Moreover, spelling correction was applied using the open source spell checker Jazzy[9].

They made heavy use of seven sentiment lexicons: AFINN (Nielsen 2011), General Inquirer (Stone et al. 1966), Bing Liu's Opinion Lexicon[10], MPQA Subjectivity Lexicon (Wilson et al. 2005), NRC Hashtag Sentiment Lexicon (Mohammad et al. 2013), Sentiment140 Lexicon (Mohammad et al. 2013) and SentiWordNet (Baccianella et al. 2010). In addition, they categorized the lexicons into the categories FORMAL and INFORMAL. A lexicon is considered INFORMAL when it includes words with spelling errors which is the case for the automatically generated twitter specific lexicons. The categorization is shown in table 3.5.

Category	**Lexicons**
FORMAL	General Inquirer, MPQA Subjectivity Lexicon, SentiWordnet
INFORMAL	AFINN-111, Bing Liu's Opinion Lexicon, NRC Hashtag Sentiment Lexicon, Sentiment140 Lexicon

Table 3.5: Categorization of sentiment lexicons by Miura et al. (2014).

POS tagging and lemmatization for lexicon lookup was done with different POS taggers for the FORMAL and INFORMAL categories. For the FORMAL category they used the Stanford POS Tagger (Toutanova et al. 2003). Lookups for the INFORMAL category were done with CMU ARK POS Tagger (Owoputi et al. 2013). Directly after POS tagging, negation annotation was performed.

For each sentiment (positive, negative) for every lexicon they implemented the following features: count of total tokens matched, total sentiment score, maximum sentiment score and the score of the last word in the tweet. This resulted in 56 lexicon based features.

Additionally, they performed word sense disambiguation on the lemmas generated with the Standford POS Tagger. Word sense disambiguation is the task of deciding which sense or meaning of a word is used in a given sentence, when the word has more than one meaning. They used UKB[11] which implements the method proposed by Agirre and Soroa (2009). The results of the disambiguator are used as features weighted with their score.

[9] `http://jazzy.sourceforge.net`, last checked 10.09.2015.
[10] `http://www.cs.uis.edu/~liub/FBS/sentiment-analysis.html`, last checked on 10.09.2015.
[11] `http://ixa2.si.ehy.es/ukb`, last checked 09.10.2015.

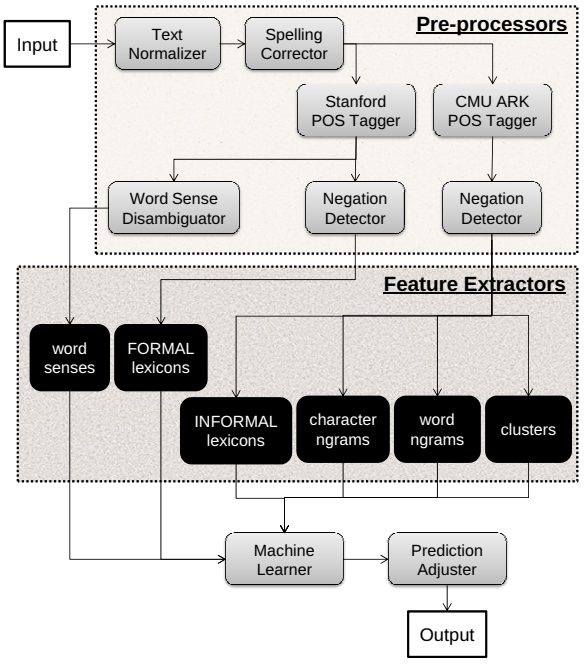


Figure 3.1: Overview of the TeamX system, taken from (Miura et al. 2014)

They used the CMU ARK POS Tagger to extract n-gram features and also used the word cluster features provided by the tagger.

An overview of the system is shown in figure 3.1.

As machine learning algorithm they used Logistic Regression, namely the implementation from LIBLINEAR (Fan et al. 2008). Parameters were optimized using a standard grid search. To bias the prediction according to the general polarity distribution of tweets they introduced a weighting factor w_l for each polarity l that adjusts the probability output $Pr(l)$ of the classifier. The prediction label is selected according to the following equation:

$$\underset{l \in \{positive, negative, neutral\}}{\arg\max} \quad score(l) = w_l Pr(l) \tag{3.2}$$

Miura et al. (2014) used different POS taggers for the FORMAL and INFORMAL lexical ressources and also introduced a bias to their classifier which helped to model the polarity distribution across tweets.

3.3.2 Coooolll: A deep Learning System

One of the two systems which harnessed deep learning methods is "Coooolll: A Deep Learning System for Twitter Sentiment Classification "(Tang et al. 2014a). While most of the other participants focused on the design of effective features, Coooolll made use of continuous representations of words, also known as word embeddings[12].

[12]See (Mikolov et al. 2013a) or section 2.1.2 for further details.

Team Coooolll created a neural network architecture capable of learning sentiment-specific word embeddings (SSWE) (Tang et al. 2014b). Their architecture is an extension of the well known C&W model (Collobert et al. 2011). The basic idea of the model is to obtain continuous vectors capturing the distributional semantics of words based on their syntactic context. A neural network is trained along with a lookup table including the word vectors. The network predicts a single output score f^{cw}. Training the network is done by computing the score of a n-gram t from training corpus and then replacing one of the tokens with a random token, creating a corrupted n-gram t^r. The loss function used can then be defined as:

$$loss_{cw}(t, t^r) = max(0, 1 - f^{cw}(t) + f^{cw}(t^r)) \tag{3.3}$$

This procedure trains the network to assign higher scores to the correct n-grams than to the corrupted n-grams. The actual training process can be done using the backpropagation algorithm. Team Coooolll extended this architecture by letting their system $SSWE_u$ predict a tuple (f^{cw}, f^s), where f^s represents a sentiment score. They used emoticons as noisy labels to label the training data with a sentiment and created a second loss function which forces the network to assign higher sentiment scores to the correct n-grams than to the corrupted ones:

$$loss_s(t, t^r) = max(0, 1 - \delta_s(t) f^s(t) + \delta_s(t) f^s(t^r)) \tag{3.4}$$

where $\delta_s(t)$ indicates the sentiment of the current training tweet's polarity, with 1 meaning *positive* and -1 meaning *negative* . Both loss functions are then combined to a final loss function with a control parameter α to tune the influence of the loss functions:

$$loss_s(t, t^r) = \alpha \cdot loss_{cw}(t, t^r) + (1 - \alpha) \cdot loss_s(t, t^r) \tag{3.5}$$

Finally, the authors explored various types of convolution layers (Collobert et al. 2011, Mitchell and Lapata 2010, Socher et al. 2011) to obtain a fixed dimension vector representation for a tweet resulting in the concatenation of multiple convolution layer results.

Along with the SSWE features described above the authors used a set of STATE features, mainly the hand-crafted feature sets suggested by Mohammad et al. (2013).

An overview of the process is shown in figure 3.2.

They used the training data provided by SemEval to train a SVM classifier using LIBLINEAR (Fan et al. 2008) with the parameters (*-c, -wi*) optimized on the 2013 test set.

Team Coooolll showed that word embeddings as features for message level polarity classification of tweets perform just as well as other state-of-the art methods and yield even better results when combined with them. Moreover, their results indicate that domain specific word embeddings outperform general purpose word embeddings by a large margin.

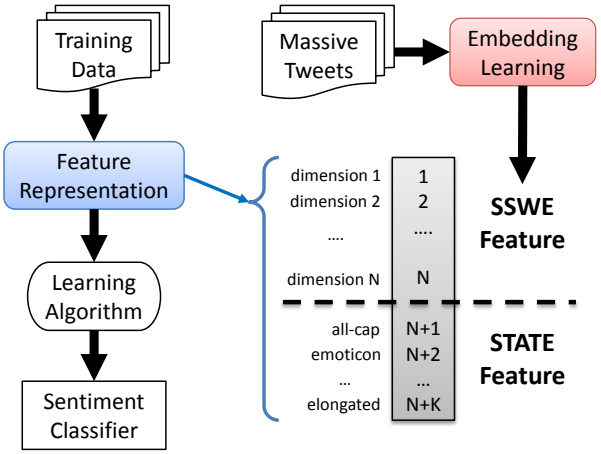


Figure 3.2: Overview of the Coooolll system, taken from (Tang et al. 2014a)

3.3.3 RTRGO: Enhancing Rich Features with Stochastic Gradient Descent

Team RTRGO (Günther et al. 2014) ranked third and is based on the authors' 2013 contribution *GU-MLT-LT* (Günther and Furrer 2013). Basically, they added some additional surface features, made use of lexical resources and incorporated feature weighting and random subspace learning into their approach.

For tokenization they used the Ark tokenizer (Owoputi et al. 2013). Normalization of the tokens was performed to deal with lexical sparsity. All tokens were converted to lowercase and the # character was removed from hashtags. Tokens, which could not be found in word lists or the used sentiment lexica, were normalized by removing repeated letters (goooooooood -> goood -> good) until the there were no more letters to be removed or the token was recognized by one of the aforementioned resources.

They made use of the following features:

- Unigrams and bigrams (stopwords and punctuation removed from bigrams) were used in a bag-of-words manner.
- Word stems were generated by a Porter stemmer (Porter 1980).
- IDs of the tokens' word clusters (Owoputi et al. 2013) were used.
- Presence of hashtags or URLS was noted.
- Two features were used for capturing if the majority of tokens was in a negative or positive sentiment list for each of the following lexicons: Bing Liu (Hu and Liu 2004), MPQA subjectivity lexicon (Wiebe et al. 2005) and the NRC hashtag lexicon (Mohammad et al. 2013).
- All tokens, stems, word clusters and lexical features were computed using standard negation handling.
- Names and user mentions were removed.

- Feature weighting was applied: The default weight for all features is 1, it was set to 2 if a token was all uppercase, had more than three adjacent repetitions of a character or if the token was an adjective or emoticon. Tokens were weighted with 0.5 when they occurred in a question context.

The machine learning method used was the same as in 2013, linear one-vs-all classifiers trained with stochastic gradient descent using hinge loss and elastic net regularization, as implemented in *scikit-learn* (Pedregosa et al. 2011).

Noting that training on lexical features can yield brittle NLP systems which are prone to overfitting, the authors applied the method of *random subspace learning* (RSSL) (Søgaard and Johannsen 2012). Basically, RSSL forces the learning algorithm to produce a model with increased redundancy, by training the model K times while leaving out a fixed percentage of features for each $k \in K$. Finally, all K models are averaged to generate a single more robust model for classification.

Moreover, an ablation study for the proposed feature groups was performed. Its results are shown in table 3.6.

Feature Set	**Avg.** F_1
ALL	69.08
- bigrams	-0.31
- stemming	-0.92
- word clusters	-1.29
- lexical features	-1.99
- negation handling	-1.43
- name exclusion	+0.08
- feature weighting	-0.73
- RSSL	-0.67

Table 3.6: Feature ablation study results from Günther et al. (2014)

As was the case with most approaches, the lexical features have the biggest performance impact, followed explicit negation handling and word clusters. Surprisingly, the exclusion of named entities had a very small but negative impact on the results. The novel ideas of tweet specific feature weighting based on the token's surface structure and the random subspace learning yielded notable improvements.

3.3.4 NRC-Canada-2014: Recent Improvements of the Usage of Sentiment Lexicons

NRC-Canada, winner of SemEval 2013 (Mohammad et al. 2013), improved their system and achieved competitive results in 2014 (Zhu et al. 2014), ranking fourth. The main difference of their approach compared to the 2013 version is the handling of negation.

The authors (Kiritchenko et al. 2014) state:

> "A word in a negated context has a different evaluative nature than the same word in an affirmative (non-negated) context. This difference may include the change in the polarity category (positive becomes negative or vice versa), the evaluative intensity, or both. For example, highly positive words (e.g., "great") when negated tend to experience both, polarity change and intensity decrease, forming mildly negative phrases (e.g., "not great"). On the other hand, many strong negative words (e.g., "terrible") when negated keep their negative polarity and just shift their intensity. The conventional approach of reversing polarity is not able to handle these cases properly."

To capture the aforementioned information in the prior knowledge base of their system, they recreated their automatically generated sentiment lexicons from 2013 (Mohammad et al. 2013). The tweet corpus was split into two parts, the *Affirmative Context Corpus* and the *Negated Context Corpus*. Negated parts of a tweet went into the negated context corpus while the non-negated parts went into the affirmative context corpus while the polarity labels of parts stayed the same as the tweets they originated from. Otherwise, the two lexicons were created as in the 2013 approach. For further details see Mohammad et al. (2013), Kiritchenko et al. (2014).

Table 3.7 shows some example scores in comparison to the lexicon without negated context treatment.

Term	**Lexicons**		
	Base	**AffLex**	**NegLex**
Positive terms			
great	1.177	1.273	-0.367
beautiful	1.049	1.112	0.217
nice	0.974	1.149	-0.912
good	0.825	1.167	-1.414
honest	0.391	0.431	-0.123
Negative terms			
terrible	-1.766	-1.850	-0.890
shame	-1.457	-1.548	-0.722
bad	-1.297	-1.674	0.021
ugly	-0.899	-0.964	-0.772
negative	-0.090	-0.261	0.389

Table 3.7: Sentiment score example from the base lexicon without context handling (Base), the Affirmative Context (AffLex) and Negated Context (NegLex) Lexicons, taken from Kiritchenko et al. (2014).

These results strengthen the hypothesis of Kiritchenko et al. (2014). Positive terms such as "great" are assigned a higher score in the affirmative context lexicon because the data is no longer polluted with negated appearances of the term. Most of the positive terms

flip their polarity in a negated context such as "not good", resulting in a negative score in the negative context lexicon. Negative terms on the other hand tend to more rarely flip their polarity but loose strength in negated contexts. Consider for example the phrase "not that terrible" where "terrible" appears in a negated context but still carries negative sentiment.

By splitting their automatically generated sentiment lexicons into an affirmative context lexicon and a negated context lexicon the authors managed to significantly increase the performance of their system by handling negation in a very advanced manner.

3.3.5 TUGAS: Exploiting Unlabeled Data

Team TUGAS (Amir et al. 2014) ranked fifth with their system focused on exploiting the usage of unlabeled external data.

Their preprocessing consisted of lower casing, replacement of user names and URLs with special tokens, normalization of elongated words (goooooood -> goood) and negation annotation as suggested by Pang et al. (2002).

Feature sets investigated by the authors consisted of unigram bag-of-word features, Brown Clusters as provided by Owoputi et al. (2013), Concise Semantic Analysis (Li et al. 2011), Dense Word Vectors as proposed in Mikolov et al. (2013b) and obtained by using *word2vec*[13] on a 17 million tweet corpus. Moreover, they incorporated lexicon features as proposed in Mohammad et al. (2013) and syntactic features such as the number of elongated words, number of sequences of certain punctuation, the number of positive and negative emoticons and the number of capitalized words.

Both the unigram bag-of-words and the Brown Cluster features were weighted with the corresponding term's $\Delta BM25$ heuristic (Paltoglou and Thelwall 2010) which can be computed as:

$$\Delta BM25(w_i) = tf_i \times \log\left(\frac{(N_p - df_{i,p} + s) \cdot df_{i,n} + s}{(N_n - df_{i,n} + s) \cdot df_{i,p} + s}\right) \tag{3.6}$$

where tf_i is the frequency of term i in the message, N_a is the size of corpus a with $a \in p, n$ (positive, negative), $df_{i,a}$ is the document frequency of term i in corpus a (the number of documents it occurs in) and s is a smoothing parameter set to 0.5 by the authors.

The dense word vector representation of a tweet m was computed as the sum of the vector representations w_j for all terms present:

$$m_{vec} = \sum_{j \in m} w_j \tag{3.7}$$

[13] See `https://code.google.com/p/word2vec/`, last checked 15.09.2015.

Moreover, the authors created a *polarity vector* p_c for each class c as follows:

$$p_c = \frac{1}{N_c} \sum_{m \in c} m_{vec} \tag{3.8}$$

where m is a message of the class c, and N_c is the total number of messages in class c. These vectors are basically the center of each class in the vector space and can be seen as the prototypical representation of that class. Features generated from these vectors were the euclidean distance to all classes' polarity vectors.

For classification they used a L2-regularized logistic regresssion implementation from *scikit-learn* (Pedregosa et al. 2011) trained in an one-vs-all manner, as this is the default of *scikit-learn* for multi class problems.

Table 3.8 shows the results of their feature impact study:

Features	**2013 Official**	**2014 Official**
submitted	65.60	69.00
- lexicons	61.70	66.40
- classVec	65.40	69.60
- wordVec	66.0	68.00
- bow-bc	65.10	65.30
+ syntactic	65.70	68.50
+ csa	60.50	67.50
+ bow-uni	58.50	66.70

Table 3.8: Impact of the removal/addition of the feature groups, taken from Amir et al. (2014)

The features listed are: lexicon features (lexicon), polarity vectors (classVec), dense word vectors (wordVec), Brown Clusters (bow-bc), syntactic features (syntactic), Concise Semantic Analysis (csa) and unigram bag-of-words (bow-uni). The score shown is the official evaluation score of the contest, macro F_1-Score of the positive/negative classes. The row labeled *submitted* denotes the system submitted by the authors, including lexicons, classVec, wordVec and bow-bc.

Adding syntactic, csa and bow-uni features actually led to drops in performance. Lexicons, wordVec and bow-bc features led to a noticeable improvement on both tests. The classVec features yield a small improvement on the 2013 data but had negative impact on the 2014 test set.

All in all, the authors harnessed the power of unlabeled resources such as dense word vectors and word clusters and combined those with intelligent feature weighting for the bag-of-word models and state-of-the-art lexical features.

3.4 SemEval 2015

The popular task "Sentiment Analysis in Twitter" which was successfully run in 2013 (Nakov et al. 2013) and 2014 (Rosenthal et al. 2014) was continued in 2015 (Rosenthal et al. 2015). Additionally to the tasks A and B which were rerun in 2015, three new tasks C, D and E were introduced which focused on topic-based message polarity, detecting trend towards a topic and determining strength of association of Twitter terms with positive sentiment. Since this thesis focuses on message level polarity classification, only the results of subtask B are discussed here.

Scoring of the submissions was done as it was in 2013 and 2014 but a new test dataset for 2015 was created which included additional sarcastic tweets identified by the labelers but which was constructed in the same manner as before otherwise.

Most of the approaches continued to be supervised machine learning methods harnessing sentiment lexicons, syntactic and surface features and bag-of-words. The presence of word embeddings or deep learning approaches was further strengthened.

3.4.1 Webis: Ensemble of Top Ranking Systems

Team Webis ranked first in Subtask B of SemEval 2015 with their approach "Webis: An Ensemble for Twitter Sentiment Detection" (Hagen et al. 2015). The basic idea of the approach is to select well performing and preferably diverse systems from SemEval 2013 and 2014 and combine them using an ensemble approach that is based on the single classifiers' confidence scores.

They picked the best performing system of 2013 - NRC-Canda (Mohammad et al. 2013) - and then chose other approaches which are significantly dissimilar from NRC-Canada. The other chosen systems from 2013 were GU-MLT-LT (Günther et al. 2014) and KLUE (Proisl et al. 2013). From 2014 they used the TeamX system (Miura et al. 2014). While reimplementing the systems, Hagen et al. (2015) noticed that some information or data was missing and they could not reimplement the systems exactly as they were originally proposed. However, Hagen et al. (2015) argue:

> "If an approach can be reimplemented with incomplete information and if it then achieves a performance within the ballpark of the original, it can be considered much more robust than an approach that must be precisely the same as the original to achieve its expected performance."

Some of the design decisions they had to make due to lack of information even led to increased performance in their reimplemented version of the systems.

While standard ensemble strategies such as bagging (Breiman 1996) and boosting (Schapire 1990) yielded no satisfactory results, the authors resorted to a much simpler strategy that performed significantly better. They just averaged the confidence scores of the four classifiers to obtain an overall result. This strategy outperformed other strategies such as majority votes. Consider for example that two classifiers might be very sure of a tweet being positive while the other two vote for negative but with

little confidence. Majority voting would lead to a draw while averaging produces a probably more reliable result.

All in all, the combination of well performing diverse systems to suitable ensemble methods seems to be a promising idea which has to be further investigated.

3.4.2 UNITN: Deep Convolutional Neural Network

Rank two of the 2015 message polarity task was taken by team UNITN (Severyn and Moschitti 2015). They describe a deep convolutional neural network for classification with their main contribution being the initialization process of the network's parameters.

The network's architecture is based on earlier approaches which were developed to perform various sentence classification tasks (Kalchbrenner et al. 2014, Kim 2014). It consists of a single convolutional layer, a max-pooling layer as non-linearity and a softmax classification layer. While Kalchbrenner et al. (2014) used a sequence of multiple convolution layers, Severyn and Moschitti (2015) chose to use only one layer due to computational efficiency, since the results of Kim (2014) suggest similar performance to multiple layers. Activiation units used after the convolution layer consisted of a rectified linear function defined as $max(0, x)$, as it speeds up training due to its computational simplicity and in some cases even yields better results than the more common sigmoid or hyperbolic tangent functions (Nair and Hinton 2010).

Figure 3.3 provides an overview of the architecture.

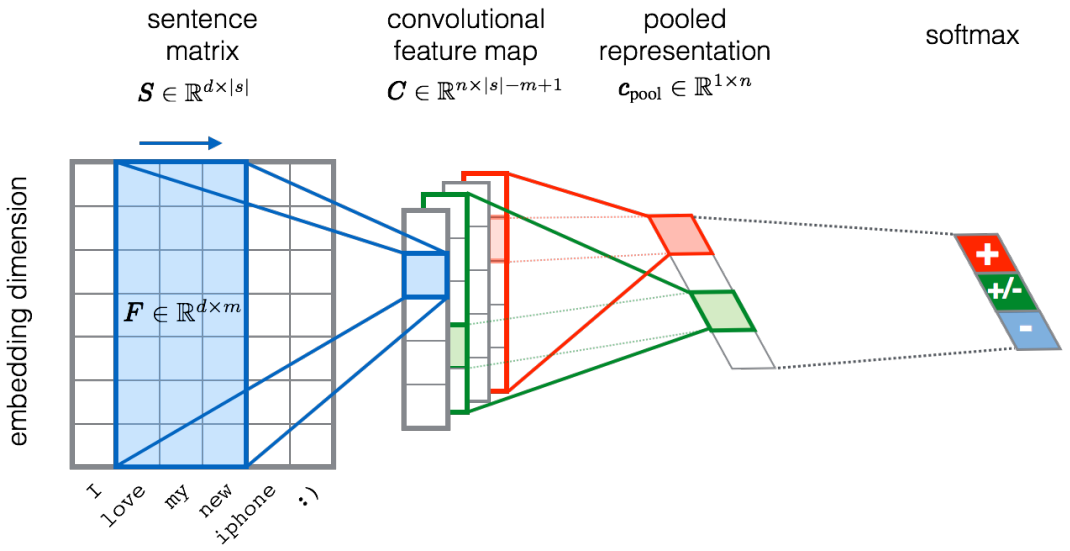


Figure 3.3: Overview of the UNITN deep neural network architecture, taken from Severyn and Moschitti (2015). The symbol d denotes the dimension of the word embeddings, m is the width of the convolution filter and n is the number of convolutional feature maps.

Input of the system is a sentence matrix S which is generated by looking up the words present in the tweet in a lookup table which includes word embeddings for each word

in the vocabulary.[14]

The novelty of the approach lies in the initialization of the word embeddings. Usually, the word embeddings are learned along with the normal training process of the network. However, this is only possible with sufficient amounts of training data and the relatively low number of tweets provided as a training set from SemEval are probably not sufficient. Hence, the authors developed a three step approach for training the network:

1. The authors used the *word2vec* neural language model (Mikolov et al. 2013b) to learn initial word embeddings of high quality. They collected 50M random tweets over a period of two month. They applied minimal preprocessing to tweets by tokenizing them and normalizing URLs and user names. Parameters used for training the word embeddings with *word2vec* were the skipgram model with a window size of 5. Moreover, tokens which occurred less than 5 times were ignored.

2. General purpose word embeddings such as the ones generated in step 1 capture important semantic and syntactic aspects of the words but contain no information about sentiment. To further improve the word embeddings the authors used a distant supervision approach similar to Go et al. (2009) to collect 10M tweets which were labeled with a sentiment according to the emoticons present in the tweet. The word embeddings from step 1 were refined using the acquired 10M tweets to train the network.

3. As a last step, the network and its parameters generated by step 2 were used as a starting point to initialize the system for training with the manually annotated tweets provided as training data by SemEval.

Table 3.9 shows the results of a comparison between the initialization methods.

Dataset	**Random**	**Unsup.**	**Distant**
Twitter'13	64.51	72.35	72.79
Twitter'14	63.69	71.07	73.60
Sarcasm'14	46.10	52.56	55.44

Table 3.9: Results of the models performance on the SemEval-2015 progress dataset using different initialization methods: *Random* (random word embeddings), *Unsup.* (*word2vec* embeddings) and *Distant* (pretrained with the distant supervised data), taken from Severyn and Moschitti (2015).

The results indicate that general purpose word embeddings seem to outperform randomly initialized word embeddings and refinement with distantly supervised, domain specific word embeddings improves the results even further.

[14]For a more in-depth explanation of the layer types see section 2.2.4 or Severyn and Moschitti (2015).

The authors demonstrated the efficiency of a new, distantly supervised initialization strategy for a deep convolutional neural network resulting in a system which performs on par with other state of the art systems while not relying on any handcrafted features or extensive preprocessing of the data.

3.4.3 Lsislif: Feature Extraction and Label Weighting

Team Lsislif (Hamdan et al. 2015) ranked third with their approach consisting of a logistic regression classifier trained with various feature groups including n-grams, lexicon features, semantic features and statistical measures such as the Z-score.

They preprocessed their data by replacing slang words and abbreviations with their more common expressions, for example *gr8 -> great*. Moreover, they performed negation handling by annotating words in a negated context (delimited by punctuation) with a negation suffix such as *_NEG*. Unigram and bigram features were then created, ignoring tokens which occurred less than three times.

Lexicon features were based on two manual constructed lexicons (Hu and Liu 2004, Wilson et al. 2005) and three automatically generated lexicons (Mohammad et al. 2013, Baccianella et al. 2010). For all lexicons they computed the number of positive words, the number of negative words, the number of positive words divided by the number of negative words and the polarity of the last word. In addition, for the automatically created lexicons they computed the sum of the positive scores and the sum of the negative scores.

Semantic features consisted of the commonly used 1,000 Brown clusters provided by the CMU ARK group (Owoputi et al. 2013) and 10 topic features created by Hamdan et al. (2015) using *latent Dirichlet association* on the training data provided by SemEval.

Furthermore, a statistical measure was used - the Z-score. The purpose of the Z-score is to measure how many standard deviations an element deviates from the mean. Hamdan et al. (2015) used it to compute how much a terms frequency for a particular class deviates from the mean occurrence of that term in the whole corpus. Using this statistical value they computed their features as the number of words having a score higher than a chosen threshold for each of the three classes, and the class having the minimum and maximum number of words with a score higher than the chosen threshold.

Another feature used was *semantic role labeling* as proposed by Ruppenhofer and Rehbein (2012) which assigns tokens a label according to their semantic role in the sentence.

Assessing their proposed feature groups with an ablation studyjj(one feature group left out at a time) and an incremental addition study (adding one feature group at a time to the base line) they found that lexicon features had the strongest impact. The Brown clusters also yielded consistent improvements, so did the newly created topic clusters. However, the proposed Z-score based features yielded conflicting results, sometimes they increased performance, sometimes they decreased it. Similar behaviour could be observed for the semantic role abeling features.

All in all, the authors confirmed the results of previous work in the field that lexicon features have a strong impact on performance. Additionally, they suggested new features groups to be used of which semantic role labeling and Z-score did not perform consistently well and only the created topic cluster yielded notable benefits.

3.4.4 INESC-ID: Embedding Subspaces without Hand-Coded Features

Team INESC-ID (Astudillo et al. 2015) ranked fourth with their approach. They presented a model without any hand-coded features that does not make use of linguistic resources. The basic idea of the approach is to project pre-trained general purpose word embeddings into a small subspace relevant to the sentiment classification task and estimate the classes' probabilities using a non-linear model.

Basically, they used the popular *word2vec* model (Mikolov et al. 2013b) but incorporated a modification proposed by Ling et al. (2015). Instead of the skip-gram model used by Mikolov et al. (2013b), they used a so called *structured skip-gram* model. The main difference is that the structured skip-gram model does not only consider the presence of words in the context but also their order. Hence, the resulting word embeddings capture syntax features better as was shown in Ling et al. (2015). They used the 52 million tweet corpus that was used in (Owoputi et al. 2013) for training of the word embeddings.

Astudillo et al. (2015) state that "[i]deally, embeddings should be adapted to the supervised task [...]". However, given the presence of only a small subset of word embeddings in the training data, words not present will never have their embeddings updated. So, the authors proposed a projection scheme where instead of just using words embeddings $\mathbf{E} \in \mathbb{R}^{e \times v}$ (e is the embedding dimension and v is the vocabulary size) they use the adapted embeddings $\mathbf{S} \cdot \mathbf{E}$, where $\mathbf{S} \in \mathbb{R}^{s \times e}$ (s is a parameter of the system to control the size of the projection) with $s << e$. The matrix $\mathbf{S}$ is trained on the supervised data and is used to project the word embeddings to a subspace suitable for the sentiment classification task. The hope is that this compensates the problem that not all word embeddings are updated, by creating a subspace mapping which influences all embeddings and hopefully generalizes well. Words occurring in the test data which are not in the embedding vocabulary are represented by an embedding vector of only zeros.

The non-linear subspace model they built estimates the probability of each possible class $y = k$ as:

$$p(y = k|\mathbf{m}; \mathbf{C}, \mathbf{S}) = exp(\mathbf{C}_k \cdot \sigma(\mathbf{S} \cdot \mathbf{E} \cdot \mathbf{m}) \cdot \mathbf{B}), \tag{3.9}$$

where $\mathbf{m} \in \{0, 1\}^{v \times n}$ is a matrix representing a tweet composed of n words by including a one-hot vector (only the words index has a 1, the rest is 0) for each word present in the tweet. The function $\sigma()$ is the sigmoid function applied element-wise to the argument matrix. The matrix $\mathbf{C} \in \mathbb{R}^{3 \times s}$ maps the embedding sub-space to the classification space and the matrix $\mathbf{B} \in 1^{n \times 1}$ sums the scores of all words before the normalization by the $exp()$ function. Basically, the model is equivalent to a multi-layer perceptron (Rumelhart et al. 1985) with one hidden sigmoid layer and a soft-max output layer.

Training was done by minimizing the negative log-likelihood of the correct class using *Stochastic Gradient Descent* (Rumelhart et al. 1985) with a learning rate of 0.01.

Moreover, the authors investigated embedding sizes of 50, 200, 400 and 600 and sub-space dimensions of 5, 10, 20 and 30. The best performance on the 2015 SemEval test set was achieved with an embedding dimension of 600 and a sub-space size of 10.

All in all, the authors used word embeddings tuned for syntax and used a bag-of-word representation (in this case each word is represented as a word embedding) of the tweets as input for a non-linear sub-space model that is equivalent to a multi-layer perceptron. They did not rely on any hand-crafted features or lexical resources and performed only minimal preprocessing.

3.4.5 Splusplus: Feature-Rich Two-Stage Classifier

Team Splusplus Dong et al. (2015) used a two stage approach to predict the labels of the given tweets. They performed a subjective/objective classification and a positive/negative classification on the subjective tweets. Furthermore, they proposed a *polarity boosting trick* to incorporate the classification scores of the positive/negative classification into the subjective/objective classification step.

Preprocessing of the data included normalization by replacing user mentions and URLs with wild card tokens. Moreover, elongated words were reduced to a maximum character repetition of three.

Commonly used basic features such as word and character level n-grams, skip-grams, brown clusters (Owoputi et al. 2013), presence or absence of POS, lexicon features as suggested by Mohammad et al. (2013) and Twitter-specific surface form features such as the presence of hashtags, and the number of elongated words, emoticons and punctuations.

Besides, various features based on deep learning were incorporated. The authors build 40-dimensional *word2vec* (Mikolov et al. 2013b) word embeddings from a tweet corpus. The resulting word embeddings were clustered into 4,960 clusters using k-means clustering with L2 distance. Presence/absence of words from the clusters are used as features for the subjective/objective classification only. A neural network architecture as proposed by Collobert et al. (2011) was used to generate features. The architecture consists of a convolution layer, a max pooling layer, a hidden layer and a softmax layer. The output of the softmax layer - a three dimensional vector containing the posterior probability estimates for each of three classes - is used as feature. Finally, the sentiment-specific word embeddings by Tang et al. (2014a) were used to compute features by using element-wise max, min and avg operations.

The authors showed that a two stage approach for classification may open up new possibilities to engineer different features for subjective/objective classification and positive/negative classification. Moreover, they demonstrated in a feature ablation study that their proposed features, derived from various deep learning methods, yield notable improvements over using just basic features.

3.5 SemEval 2016

In 2016 the task of message level polarity classification (subtask A) (Nakov et al. 2016) for tweets was still popular. Datasets of the previous years were provided as test and development data and a new dataset was created for training and testing. For the message level polarity task the evaluation procedure is the same as in the previous installments of the contest. Most of the top ranking approaches are centered around the usage of word embeddings and deep neural networks. Usage of hand coded surface feature declined compared to the previous years.

3.5.1 SwissCheese: CNN Ensemble with Distant Supervision

Team *SwissCheese* entered the competition with an ensemble of convolutional neural networks that make use of large amounts of data by harnessing distant supervision (Deriu et al. 2016).

The general neural network architecture is based upon the work of Severyn and Moschitti (2015)[15] but consists of two CNN layers instead of one[16]. A schematic representation of the architecture is shown in figure 3.4.

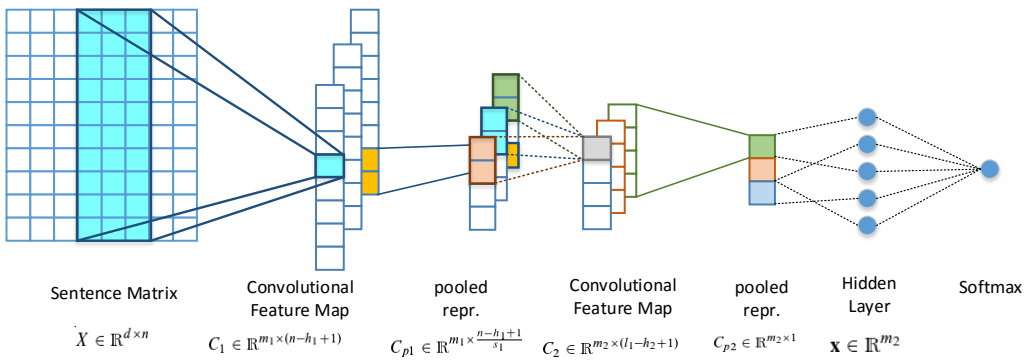


Figure 3.4: Overview of the SwissCheese deep neural network architecture, taken from Deriu et al. (2016).

Intending improved generalization the authors trained two such systems with different training procedures and datasets.

System I was initialized with unsupervised word embeddings created by word2vec (Mikolov et al. 2013a), pretrained with 90 million distantly supervised tweets[17] and finally trained with the training data provided by SemEval-2016.

System II was initialized with word embeddings created by GloVe (Pennington et al. 2014), based on a different corpus than system I. Additionally, some surface features have been encoded within the word vectors such as elongated words or punctuations

[15] See section 3.4.2 for a summary.
[16] The authors refer to a *layer* as the combination of convolutional layer and a pooling layer.
[17] Emoticons were used as a label, as suggested by Go et al. (2009).

that repeated more than three times. System II was then also trained with distantly supervised data and the training data from SemEval-2016 but using slightly different system parameters.

A meta-classifier (random forest) was built using the three softmax outputs and the resulting categorical value of both systems as input.

The System ranked 1st with an F-score of 63.30 on the 2016 test set.

3.5.2 SENSEI-LIF: Polarity Embedding Fusion for Robust Sentiment Analysis

Team SENSEI-LIF (Rouvier and Favre 2016) applies an approach originally developed for the European FP7 project SENSEI[18] to the message level polarity classification task of SemEval. The system is an extension of the CNN architecture introduced by Poria et al. (2015) and makes use of additional lexical information as proposed by Ebert et al. (2015).

In a first step three CNNs are created. Each of the systems uses a different target function for generating the underlying word embeddings. *Lexical embeddings* are created with the classical skip-gram model (Mikolov et al. 2013a) by having the model predict for a given context window $w_{i-2} \dots w_{i+2}$ the output $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$ with w_i as input. *Part-of-speech* embeddings are generated by creating joint embeddings for the tokens and their part of speech. Given a context of words $w_{i-2} \dots w_{i+2}$ and their respective part of speech $p_{i-2} \dots p_{i+2}$, the model has to predict $(w_{i-2}, p_{i-2}), (w_{i-1}, p_{i-1}), (w_{i+1}, p_{i+1})$, (w_{i+2}, p_{i+2}) when given the input (w_i, p_i). *Sentiment embeddings* are created by jointly predicting words with the sentiment label of the whole sentence. Let s be the sentiment of the sentences wherin the context window $w_{i-2} \dots w_{i+2}$ appeared. Given w_i the model is trained to predict $(w_{i-2}, s), (w_{i-1}, s), (w_{i+1}, s), (w_{i+2}, s)$.

When the CNNs for each of the aforementioned variations of word embeddings are trained for the classification task, the embedding based representation of the messages is concatenated with various features based on sentiment lexicons and sentence level surface features such as the number of emoticons or the number of contiguous sequences of punctuation. The three resulting models are then combined to the final classification system, an overview is shown in figure 3.5.

Rouvier and Favre (2016) call their proposed framework for combining the models "embedding fusion". The framework does not combine the final outputs of the previously trained models but concatenates intermediate results of their hidden layers. Those concatenated vectors are then fed to a series of fully connected hidden layers which are then trained to predict the sentiment.

The system achieved an F-score of 63.00 on the 2016 test set.

[18] `http://www.sensei-conversation.eu`, last checked: 06.01.2017.

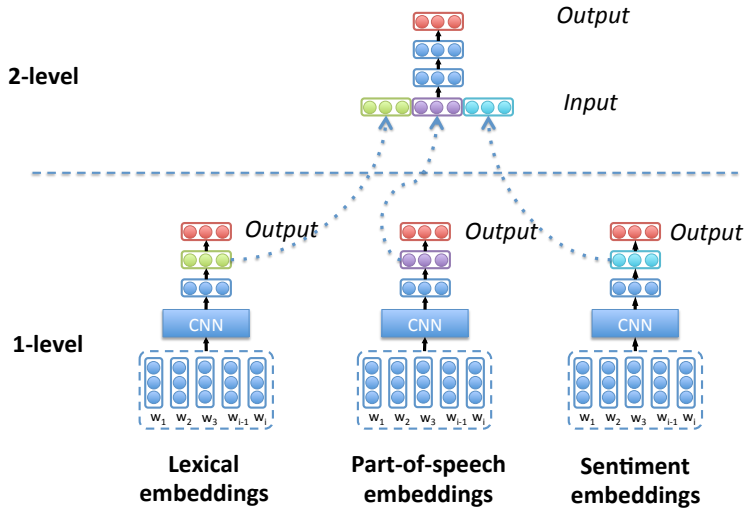


Figure 3.5: Overview of the SENSEI-LIF system, taken from Rouvier and Favre (2016).

3.5.3 UNIMELB: An ensemble of CNN, LSTM and word2vec Bayes models

Team *UNIMELB* (Xu et al. 2016) combined two kinds of neural networks, convolutional neural networks (CNN) and long-short-term-memory networks (LSTM), with a word2vec Bayes model to create an ensemble of diverse systems that outperforms most single-system architectures.

The convolutional neural network is based upon the architecture of Kim (2014)[19]. It consists of a word embedding layer, a 1-d convolution layer, a max pooling layer and a softmax layer. After initializing the word embedding layer with data generated by word2vec, the network is pretrained with distantly supervised data that uses emoticons for labeling.

LSTM networks are a variation of recurrent neural networks which were designed for handling sequential problems. Such networks store an internal state which is sophistically altered when iterating over the input sequence. One benefit of such a system is the capability to make use of the data's ordering. However, due to vanishing/exploding gradients, finding a viable architecture and training recurrent networks is considered to be a difficult task. Hochreiter and Schmidhuber (1997) proposed the LSTM as a solution to those problems[20]. The network used by Xu et al. (2016) consists of a word embedding layer, a LSTM layer and a softmax classification layer.

The word2vec Bayes model is implemented as described in Taddy (2015) with the additional inclusion of class priors. The log-likelihood is estimated via the skipgram

[19] A very similar architecture has been summarized in section 3.4.2, see also 2.2.4 for a more in-depth explanation of the layer types.

[20] Since LSTM networks are not that relevant for this thesis the details are spared here. See Xu et al. (2016), Hochreiter and Schmidhuber (1997) for further information.

objective composite likelihood[21].

All of the systems mentioned above are then combined using soft voting which Xu et al. (2016) defined as:

$$y_{\text{vote}} = \sum_i w_i y_i, \text{ s.t.} \sum_i w_i = 1, \forall i : w_i \geq 0 \tag{3.10}$$

where y_i denotes the output of classifier i. The results of the single systems and their combinations are shown in table 3.10.

Model	F-score
soft voting all	0.6000
lstm	0.5869
soft voting cnn + lstm	0.5848
cnn	0.5841
word2vec Bayes	0.4983

Table 3.10: Results for the systems and their combinations from Xu et al. (2016).

Combining all three systems outperforms all single systems and also the combination of both neural networks. While the word2vec Bayes model performs sub par on its own it seems to be essential for the overall performance gain of the ensemble.

Xu et al. (2016) ranked 3rd with a F-score of 0.617 on the 2016 test dataset.

3.5.4 INESC-ID: Reducing the Problem of Out-of-Embedding Words

Team *INESC-ID* (Amir et al. 2016) presented an extension of their contribution from 2015 (Astudillo et al. 2015)[22]. While their approach deals with the problem that pre-trained embeddings of words not occurring in the final training data are never updated, it still lacks the capability of handling out-of-embedding-vocabulary words, i.e. words occurring in the training data but not in the embedding vocabulary. Such words often occur in informal texts due to lexical variations.

According to Amir et al. (2016), a canonical solution to this problem is the usage of character-level embeddings. Instead of learning word-level embeddings e_w for a word w, embeddings e_c for the word's character-level n-grams $\{c_1, \ldots, c_m\}$ are learned. A word representation is then retrieved by combining the words character embeddings with a given combination function. A common combination function is $e_w = c_1 + \cdots + c_m$; combining the character embeddings of a words with the pointwise sum.

[21] For further details see Taddy (2015), Xu et al. (2016).
[22] A summary is presented in section 3.4.4.

However, Amir et al. (2016) found the direct usage of character-level embeddings to perform poorly. As a solution they proposed to learn an additional linear function $\mathbf{T}$ which maps the character-level embeddings into the word-level embeddings space. For all words C which occur in both vocabularies they learned $\mathbf{T}$ by solving the following optimization problem:

$$\mathbf{T} \leftarrow arg\,min_T \sum ||\mathbf{T} \cdot c_w - s_w||^2 \tag{3.11}$$

where c_w is the word representation computed from the character-level embeddings for word w and s_w is the word-level embedding for word w. The word-level embedding for an unknown word w can then be approximated by $s_w = \mathbf{T} \cdot c_w$.

Their baseline system from 2015 achieved a F-score of 0.609 on the 2016 data. Using just the character-level embeddings led to a decrease in performance to a F-score of 0.543. Incorporating the approximation for out-of-embedding-vocabulary words yielded a F-score of 0.613.

On the 2016 test set the proposed system reached a F-score of 0.610 and ranked 4th in the competition.

3.5.5 aueb: A Weighted Ensemble of SVMs

Team aueb (Giorgis et al. 2016) built an ensemble of two systems. The first system harnesses hand crafted features and is based upon Karampatsis et al. (2014). The second system relies solely on word embedding features without any manual feature engineering. An overview of the system is presented in figure 3.6.

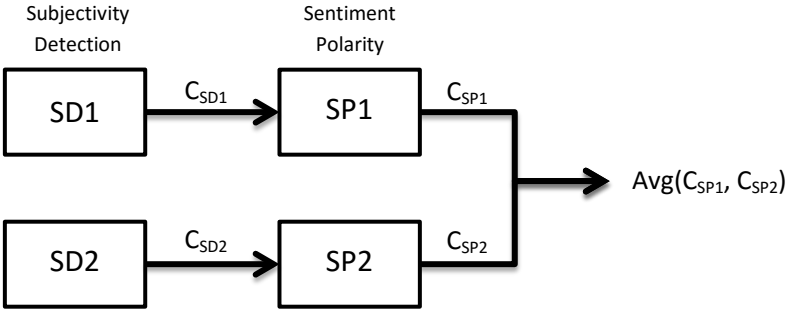


Figure 3.6: System overview of the ensemble of polarity classifiers *SP1* and *SP2*, which use the confidence of subjectivity detectors *SD1* and *SD2* as features. The image was taken from Giorgis et al. (2016).

Each of the two systems consists of two stages. In the first stage a subjective detection is performed (*SD1, SD2*). The second stage (*SP1, SP2*) uses the same feature vector representations as the first stage, with the addition of the first stage's confidence scores. Both stages use support vector machines as classification algorithm. Finally, the classification decision is made by computing a weighted average of both system's outputs.

System 1 (*SD1, SP1*) uses handcrafted features based on morphology, part-of-speech, sentiment lexicons, negation handling and topic clusters of tweets. System 2 (*SD2, SP2*) solely relies on 200 dimensional word embeddings generated with GloVe.

The authors hypothesise that the presented soft two-stage approach has certain benefits compared to a strict two-stage approach such as the one used in Karampatsis et al. (2014). While a strict approach would propagate the first stage's errors into the second stage, the soft approach incorporates the confidence signal directly into it's decision process and hence is enabled to contradict the subjectivity detection if necessary.

Moreover, the authors were able to show that their ensemble strategy outperforms the single systems on the 2016 dataset. The strict two-stage approach from Karampatsis et al. (2014) achieved a F-score of 0.5483, *SD1/SP1* reached 0.5940, *SD2/SP2* scored 0.5750 and the ensemble had a F-score of 0.6052.

3.6 Summary

The early stages of the field from 2009-2012 were dominated by distant supervision. Researchers mostly used distantly supervised tweets as training data for various machine learning algorithms. The methods differed mainly in the preprocessing methods used. Most of them used n-grams as features with few approaches harnessing additional handcrafted features. However, results were hard to compare since most researches used different datasets of unknown quality or with strong biases towards various topics or time periods.

In 2013 the SemEval competition launched their task "Sentiment Analysis in Twitter" for the first time. They created a relatively large dataset of suitable quality which was publicly available so different approaches can be compared. While the dataset is biased in various ways it was still of higher quality than anything else existing up to that point. The approaches of 2013 consisted of various supervised machine learning methods which used only the manually annotated data provided by SemEval for training. New ideas incorporated were mainly the usage of sentiment lexicons. Not only did the participants use existing, manually created lexicons but also created tweet-specific sentiment lexicons using emoticons as noisy labels. Almost all participants made use of explicit negation handling which was shown to consistently have significantly positive effects on classification performance. Moreover, various approached made use of presence or absence of words from a publicly available set of brown clusters.

SemEval 2014 reran the tasks from 2013 but added additional tweets to the test dataset, especially worthy of noting here is a set of sarcastic tweets. Approaches of the top ranking teams were mainly based on the results from 2013, making use of a set of hand crafted features in conjunction with various sentiment lexicon features. The main addition was the different incorporation of the lexicons such as the usage of different POS taggers for the general purpose and the tweet specific lexicons or the incorporation of negation handling into the automated lexicon creation process. Additionally, some teams successfully applied various term weighting schemes or improved their results by biasing their classifier according to the label distribution of tweets. One system stood

out by using custom word embeddings which were trained in a distantly supervised fashion using emoticons as noisy labels.

In 2015 three out of the top five systems made use of deep learning and/or word embeddings somehow. While two teams built deep neural networks which directly were used for classification, one team created a creative set of features derived from various deep learning methods and combined them with the commonly used basic features from 2013 and 2014. Another team managed to rank high by enhancing the basic features with additional topic clusters they created. However, the top ranking team was an ensemble created from four diverse, top ranking systems of the previous years.

In the 2016 installment of the contest, the past success of ensembles was adopted by multiple teams. Three of the five top ranking systems used ensemble strategies. Four of the five systems consisted solely or partially of deep neural networks. Only one system was an ensemble of SVMs, using - among others - word embedding features. Deep neural networks or their byproducts such as word embeddings are dominating the field.

Most of the literature from outside of SemEval is - to the best of the author's knowledge - closely related to one of the approaches that were presented above and hence is not further discussed here due to the lack of additional insights.

4 A Closer Look at the Interaction of Preprocessing and Feature Extraction for Bag-of-Word Models

Bag-of-word models have played an important role for Twitter sentiment analysis since the emergence of the field in 2009 up until today, as presented in section 3. Various combinations of preprocessing techniques and feature extraction methods have been used without further elaboration of how this exact combination was chosen and why it may be useful. A conclusive comparative study on the matter has not been performed to the best of my knowledge. That is why, Neubauer and the author started performing experiments to shed some light on the open questions in 2013. The results were published in Neubauer (2014). The study presented in this section is based upon the previous work and illuminates some of the aspects not that deeply investigated before.

4.1 Summary of Previous Results

Neubauer (2014) conducted his research at the end of the emergence of the field, parallel to SemEval 2013 and had no access to a dataset of high quality to objectively compare the preprocessing techniques and feature extraction methods of interest. Hence, a new dataset had to be created for testing performance. Since the overall goal is to assign to a tweet the sentiment the average human would assign, the ground truth has to be obtained by humans. To not only reflect the opinion of a single test subject, at least two humans have to label each tweet and only those tweets where the two labels match are used for evaluation. Moreover, the dataset should not suffer from topical bias. Some of the old datasets were collected within a few days or weeks which results in a strong bias towards events and popular topics of that time. Hence, 43 million tweets have been collected in the time from June 2012 to August 2013. About 10 million of those tweets were collected by tracking certain keywords related to product unveiling events and developer conferences from Samsung, Facebook, Sony, Microsoft and Google. Moreover, events such as the trade fairs *E3* and *CeBIT* were tracked. Another event present is the *Eurovision Song Contest*. The tracking was performed to assign higher weight to common application purposes of Twitter sentiment analysis by guaranteeing that a certain amount of relevant tweets is present. However, the remaining 33 million tweets were collected from the public twitter stream without any further filtering. From this collection of 43 million tweets, 10,176 tweets were randomly sampled and labeled by a total of 23 human test subjects. Each tweet was assigned two labels by two different test subjects. The labelers had to chose a label $l \in \{positive, negative, neutral, garbage\}$, where *neutral* represents tweets to which no sentiment could be assigned and *garbage* was to be used for messages which the labeler was not able to understand or which obviously were spam. The manually annotated data is used as test data to evaluate the different methods. For training, the distant supervision approach from Go et al. (2009)

was used. About 5 million tweets per class were collected in the aforementioned time frame. To increase the quality of the data, only tweets containing purely positive or negative emoticons and at least 5 words were included in the training set.

An overview of the two labeling stages is given in table 4.1. The column *Initial Label* shows the distribution of the first assigned label, the column *Matching Labels* presents the number of tweets which were assigned two matching labels and the column *Agreement* contains the percentage of tweets for each class on which the second labeler agreed with the first. Note that tweets initially labeled as *garbage* were no longer considered for relabeling.

Class	Initial Label	Matching Labels	Agreement
positive	1,296	910	70.2%
negative	1,711	1,328	77,6%
neutral	1,433	966	66,9%
garbage	5,726	-	-

Table 4.1: Overview of the label distribution from Neubauer (2014).

More than half of the tweets were considered *garbage* due to the labeler not being able to understand its meaning or the tweet being spam. This is in line with the findings of Tumasjan et al. (2010) who characterize most of the public Twitter stream as "pointless babble". Moreover, it seems that from the remaining data about one third of tweets has to be discarded due to labeler disagreement. The agreement percentages indicate that identification of the *negative* class seems to be easiest for humans, followed by *positive* with *neutral* having the least amount of agreement.

For the conducted experiments, only the classes *positive* and *negative* were considered.

First, Neubauer (2014) investigated the influence of feature extraction methods and preprocessing techniques on the number of features of the training corpus. Table 4.2 shows the results for the feature extraction methods.

The number of distinct features grows increasingly larger with increasing n of the n-grams. Nevertheless, the relative amount of χ^2-testable tokens is relatively stable for unigrams and bigrams but is cut in half for trigrams. The relative amount of χ^2-significant tokens is stable for all n-grams, about 20-21%. This indicates that n-grams on their own maybe quite suitable features for classification.

More surprising are the statistics for the subgrams. Subgrams were not widely used for sentiment analysis before 2013. Subgrams of size 3 show 71.8% of χ^2-testable tokens of which 51.4% occur significantly more often in one of the two classes. Subgrams of size 4 have much lower relative χ^2 counts but still seem promising.

The effects of the investigated preprocessing methods on the corpus size are presented in table 4.3.

Feature-Type	Count	χ^2-Count	χ^2-Significant
Unigrams	1,997,170	146,857 (7.4%)	32,281 (22.0%)
Bigrams	11,550,968	909,577 (7.9%)	194,869 (21.4%)
Trigrams	31,855,420	1,276,475 (4.0%)	259,095 (20.3%)
Subgrams $n = 3$	30,786	22,108 (71.8%)	11,368 (51.4%)
Subgrams $n = 4$	353,932	196,494 (55.5%)	54,999 (28.0%)

Table 4.2: Feature extraction methods and their distinct feature counts of the training corpus, features which were χ^2-testable - occurring in both classes and a minimum total of five times - and the number of features which occurred significantly more often in one class than the other. Source: Neubauer (2014).

Preprocessing	Feature Count	χ^2 Count	χ^2 Significant
None	1,997,170	146,857 (7.4%)	32,281 (22.0%)
Stopword-Removal	1,996,633 (0.027%)	146,327 (7.3%)	31,814 (21.7%)
Stemming	1,904,653 (4.632%)	127,502 (6.7%)	26,814 (21.0%)
Lemmatization	46,605 (97.666%)	27,138 (58.2%)	9,760 (36.0%)
Repeated Characters	1,952,089 (2.257%)	144,189 (7.4%)	31,669 (22.0%)
Acronym Expansion	1,996,878 (0.015%)	146,475 (7.3%)	32,022 (21.9%)
Entity Replacement	800,199 (59.933%)	111,142 (14.0%)	27,695 (24.9%)
POS-Annotation	3,444,025 (172.445%)	209,777 (6.1%)	43,681 (20.8%)

Table 4.3: Effects of preprocessing techniques on the training corpus. The column *Feature-Count* presents the number of features after applying the preprocessing step and in brackets shows the relative reduction compared to no preprocessing. All values are computed on unigrams as feature extraction method. The remaining columns denote the same data as in table 4.2. Source: Neubauer (2014).

This investigation allows for some interesting predictions with respect to the viability of the feature groups. Removal of stop words is widely used in the literature but only marginally reduces the corpus size while notably reducing the number of χ^2-significant features. Hence, one may speculate that stop words actually do carry useful information about sentiment in some cases. Stemming reduces the number of features by almost 5% but also reduces the number of valuable features. Lemmatization reduces the corpus size by 97.666%, because the lemmatizer that was used removed tokens it cannot lemmatize. However, the relative number of distinctive features is the highest of all preprocessing methods. Repeated character normalization and acronym expansion reduce the corpus size without loosing too many valuable features. Entity replacement has a notable impact on corpus size. It reduced the size to 59.933% while increasing the relative part of significant features and consequently it seems to be very promising because it drastically reduces the dimension without loosing much information. POS-annotation increases the feature count but also yields an increased number of significant features which may or may not justify the increase in dimension.

Next, a study about the amount of training data needed was performed by evaluating Naive Bayes Classifiers and Support Vector Machines using one feature type at a time.

The baseline of the results was that the maximum available amount of training data yielded the best results. For a more detailed elaboration see Neubauer (2014).

Table 4.4 summarizes the results of a feature extraction method comparison, reporting accuracy as well as precision and recall per class.

Feature Extraction	**Accuracy**	**Prec.**$_{pos}$	**Recall**$_{pos}$	**Prec.**$_{pos}$	**Recall**$_{neg}$
Unigrams$_{NB}$	80.03%	74.29%	77.80%	84.28%	81.55%
Unigrams$_{SVM}$	80.52%	73.10%	82.42%	86.80%	79.22%
Bigrams$_{NB}$	80.47%	**75.62**%	76.70%	83.88%	**83.06**%
Bigrams$_{SVM}$	**81.50**%	74.41%	**83.10**%	**87.40**%	80.42%
Trigrams $_{NB}$	77.79%	71.71%	74.95%	82.28%	79.74%
Trigrams $_{SVM}$	76.94%	69.94%	75.93%	82.48%	77.64%
Subgrams $n = 3_{NB}$	76.54%	70.37%	73.08%	81.05%	78.92%
Subgrams $n = 3_{SVM}$	75.78%	69.41%	72.31%	80.47%	78.16%
Subgrams $n = 4_{NB}$	78.60%	73.20%	74.73%	82.43%	81.25%
Subgrams $n = 4_{SVM}$	78.42%	73.18%	74.07%	82.08%	81.40%

Table 4.4: Performance of feature extraction methods when trained with maximum available corpus size. Source: Neubauer (2014).

A Support Vector Machine trained with bigrams outperforms all other configurations by about 1% accuracy. It also achieves the top scores on positive recall and negative precision. However, a Naive Bayes Classifier trained with bigrams takes the lead for positive precision and negative recall. Subgrams, while lacking in performance relatively to the other feature extraction methods, still perform reasonably well. Still, the subgrams did not perform as well as expected looking at the results presented in table 4.2. All configurations' performances are close to the baseline but the differences in precision and recall for positive and negative classes suggest a notable amount of diversity and hints at the feasibility of combining multiple combinations to ensembles.

Results of applying the various preprocessing techniques and then using unigram features for NBC and SVM are shown in table 4.5. Most of the preprocessing techniques yielded no significant change in performance. The only technique resulting in a marginal improvement was *negation annotation*. Some techniques such as *stop word removal* - as expected - led to drops in performance.

Most of the techniques investigated reduce the dimension of the feature space without significant performance drops. However, while not significant, performance often declined slightly. Improvements could only be noted in one case - *negation annotation*. Replacing or removing entities is more of a design choice to not bias the system towards certain entities, but all other techniques - with the exception of *negation annotation* - cannot be recommended.

The study of Neubauer (2014) provided some further insights into the performance of various feature extraction methods and the influence of commonly used preprocessing techniques on unigram features. Nevertheless, some questions still remain open. Is the influence of preprocessing the same for higher order n-grams, subgrams and skipgrams? How do preprocessing methods interact and influence each other when combined? Based

Preprocessing	Accuracy
Stopword Removal$_{NB}$	79.94%
Stopword Removal$_{SVM}$	79.98%
Stemming$_{NB}$	79.40%
Stemming$_{SVM}$	80.43%
Lemmatization$_{NB}$	77.30%
Lemmatization$_{SVM}$	78.33%
Repeated Characters$_{NB}$	79.98%
Repeated Characters$_{SVM}$	80.43%
Acronym Expansion$_{NB}$	79.62%
Acronym Expansion$_{SVM}$	80.38%
Entity Removal$_{NB}$	80.25%
Entity Removal$_{SVM}$	80.12%
Entity Replacement$_{NB}$	79.54%
Entity Replacement$_{SVM}$	80.07%
POS Annotation$_{NB}$	78.87%
POS Annotation$_{SVM}$	79.22%
Negation Annotation$_{NB}$	78.51%
Negation Annotation$_{SVM}$	**80.56**%
Position Annotation$_{NB}$	80.21%
Position Annotation$_{SVM}$	80.43%

Table 4.5: Effects of preprocessing techniques on unigram features. Source: (Neubauer 2014).

upon the results discussed in this section the study presented in the following sections aims to answer these questions.

4.2 Comparison of Feature Extraction Methods

In this section the influence of preprocessing on the investigated feature extraction methods is discussed. It aims to provide answers to the questions "How sensible are the preprocessing methods to the preprocessing pipeline configuration?", "Which are the best preprocessing pipelines for each feature extraction method?" and "How do feature extraction methods differ in their sensitivity to preprocessing pipelines?".

The evaluation methodology used for this experiment is basically the same as described in Neubauer (2014). Using the macro F-score as performance measure instead of accuracy is the only difference. Moreover, the same dataset is used.

The box plot in figure 4.1 summarizes the results. Each box is generated from the F-score of all pipeline combinations (about 100 per box) that were investigated for the feature extraction method the box represents. Red boxes denote the Naive Bayes Classifier's performance, blue boxes denote the Support Vector Machine's performance.

Comparing these results to the overview without preprocessing shown in table 4.4, one can observe a similar pattern. Bigrams seem to perform slightly better than unigrams in

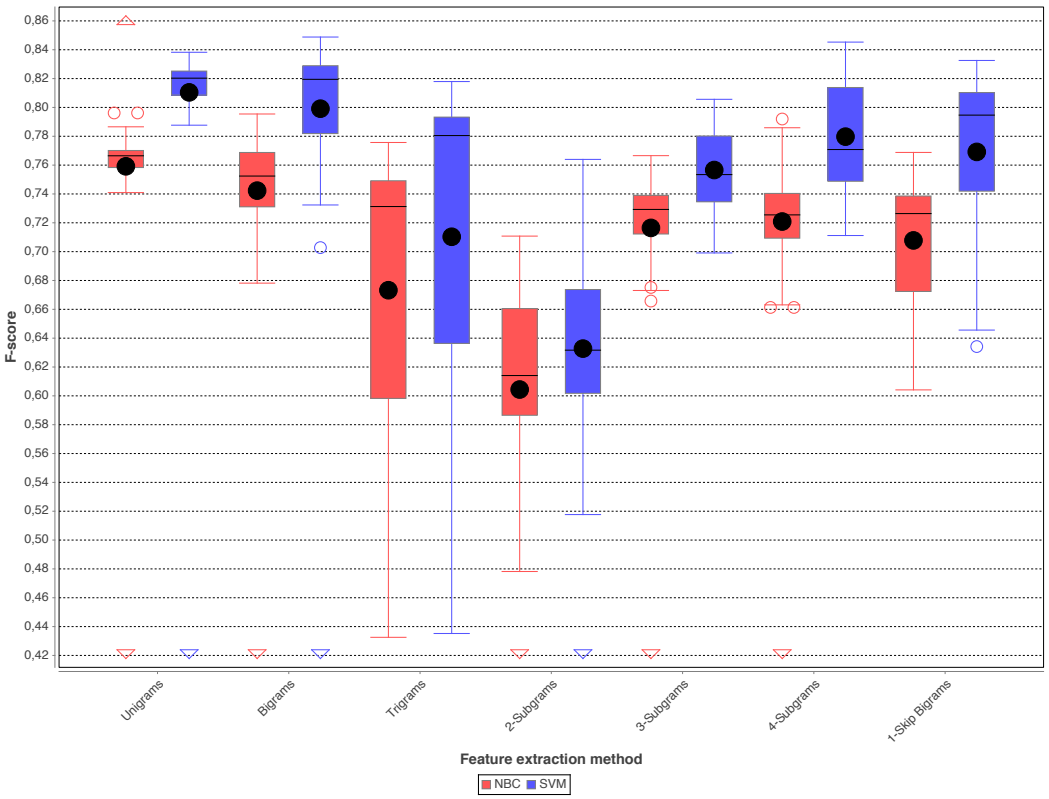


Figure 4.1: Overview comparison of feature extraction methods. The black dot denotes the mean, the black line the median and the unfilled circles and triangles denote outliers.

some cases but the median of the performance across all pipelines is almost identical with the mean being worse. Unigrams are much less sensitive to changes of the preprocessing pipeline, which manifests in a higher density of the data. Trigrams are even more sensitive than bigrams. They show worse average and median performance with the data points being scattered in a large interval. However, Trigrams perform reasonably well, in some cases but the large variance indicates that they should be used with caution and only in conjunction with the correct preprocessing. A possible explanation for the higher sensitivity could be the number of features affected by a preprocessing method. Consider a tweet consisting of five tokens `A B C D E`. A preprocessing method altering token `C` affects just one unigram (`C`), but two bigrams (`B C` and `C D`) and three trigrams (`A B C`, `B C D`, `C D E`). Hence, preprocessing techniques having positive or negative effects on all n-gram sizes seem to have their effects amplified for higher order n-grams.

The 4-subgrams seem to be able to perform competitively with bigrams in some cases but their median is much lower indicating a generally worse performance. Subgrams of size two and three perform consistently worse than all n-grams. Sensibility to the preprocessing pipeline for 3-subgrams and 4-subgrams is about the same as for bigrams. 2-subgrams show a much higher sensitivity, probably because especially preprocessing methods such as stemming or lemmatization have a very strong impact on two-character sequences.

Skipgram performance seems to consistently be slightly sub par to that of bigrams and

4-subgrams while being ahead of the other feature types. Sensitivity to the pipeline configuration is stronger than for bigrams but lesser than trigrams.

The data suggests that feature extractions methods differ strongly in their sensitivity to the choice of applied preprocessing methods when looking at the overall classification performance. In the following sections, the combinations of feature extraction methods with preprocessing techniques are highlighted and discussed in more detail.

Unigrams

The top and worst performing pipeline configurations for unigrams are shown in table 4.6. Annotating negated tokens had the strongest impact on the classification performance. Overall, it can be noted that noise-reducing preprocessing steps such as the replacement or the removal of URLs or the simple downcasing operation also seem to have positive effects. However, the absolute value of the performance improvement is only about 1% compared to the baseline of doing no preprocessing at all. As was expected, the lower ranks are filled with pipelines including POS replacement (replacing a token with its POS tag). Note, that the Performance of looking only at the POS tags present (0.710) is still far better than the majority vote baseline (0.372) and while forfeiting performance compared to the baseline of no preprocessing it may still be useful information for some use cases such as ensemble building. Looking at the lower ranks one can also discern the abundant presence of stop word removal and non-word removal (removal of tokens composed solely of punctuation or characters which are no usual word characters). Both are commonly applied in the hope of reducing noise. Nonetheless, it seems that both preprocessing techniques - removal of non-words and removal of stop words - consistently deteriorate performance. Contrary to popular belief, stop words and non-words seem to carry useful information for the sentiment analysis of tweets. When adding non-word removal and stop word removal to a well performing pipeline such as downcasing with named entity replacement (rank 39, F-score: 0.821), the result drops significantly to rank 88 with a F-score of only 0.788. Hence, preprocessing steps performing well in other domains do not necessarily work in Twitter sentiment analysis and should be considered with caution.

A surprising result is the ranking of the combination of downcasing (rank 4) and negation annotation (rank 1) which achieved only rank 2. Initially, it was expected that combinations of preprocessing techniques that perform well when used in isolation would perform better if combined. This behaviour could not consistently be observed for unigrams. Hence, one should probably not just carelessly evaluate preprocessing techniques in isolation and draw conclusions of their combination's performance but actually evaluate the combinations, since their interactions can probably not be foreseen easily.

Bigrams

Looking at the results for bigrams which are presented in table 4.7 one can notice slightly different tendencies for the top performing pipeline configurations.

Rank	Preprocessing Pipeline	F_1
1	NegationAnno.	0.838
2	Downcasing, NegationAnno.	0.836
3	URLRep., Downcasing	0.835
4	Downcasing	0.834
5	URLRem., Downcasing	0.833
6	POSAnno.	0.831
33	-	0.823
87	StopWordRem., NonWordRem., Downcasing, LemmatizeRep.	0.790
88	StopWordRem., NonWordRem., Downcasing, NamedEntityRep.	0.788
89	POSRep., NonWordRem.	0.772
90	POSRep., NonWordRem., Downcasing	0.771
91	POSRep., StopWordRem., NonWordRem., Downcasing	0.749
92	POSRep.	0.710
93	POSRep., Downcasing	0.703
94	POSRep., StopWordRem.	0.520

Table 4.6: Best and worst performing preprocessing pipelines for unigrams and the SVM classifier. The ranking is determined by ordering the pipelines with respect to the achieved F-score. Note that rank 94 is considered to be an outlier in figure 4.1 and is indicated only by the triangle below the unigram box.

A prevalence of downcasing is one of the differences coming to attention. As was indicated by table 4.2 there are about six times as many bigrams as there are unigrams in the same corpus. That is, the effect of normalization and increase in information density generated by preprocessing techniques such as downcasing seems to be magnified for bigrams, relative to unigrams, since there is not a single pipeline without downcasing in the top five bigram pipelines.

Another obviously well performing type of preprocessing is the removal or replacement of various entities such as user mentions, URLs and named entities. The top ranking pipelines consist solely of downcasing coupled with entity removal/replacement. This effect can be explained by looking closer at an example illustrating the consequences of the presence of multiple entities in a tweet. Consider the following tweet:

> "My man @jamal42123 and me like Cooper's Bar in London `http://www.tinyurl.com/123456`"

Without any preprocessing the tweet would be converted to the following n-grams: *my man, man @jamal42123, @jamal42123 and, and me, me like, like Cooper's, Cooper's Bar, Bar in, in London, London* `http://www.tinyurl.com/123456`. In total this generates 10 bigrams of which many probably occur just this one time, for example *man @jamal42123* is very unlikely to come up in another tweet. Using named entity removal, URL removal, username removal and downcasing results in the following text before tokenization:

> "my man and me like in"

Rank	Preprocessing Pipeline	F_1
1	Downcasing, NamedEntityRem.	0.849
2	Downcasing, AtMentionRem., URLRem., NamedEntityRem.	0.846
3	Downcasing, URLRem.	0.845
4	Downcasing, URLRep.	0.845
5	Downcasing, NamedEntityRep.	0.845
10	Downcasing, NegationAnno.	0.837
37	-	0.823
82	StopWordRem., PositionAnno.	0.757
83	Downcasing, StopWordRem., NonWordRem., NamedEntityRep.	0.756
84	Downcasing, StopWordRem., NonWordRem., NamedEntityRem., AtMentionRem., URLRem.	0.752
85	Downcasing, StopWordRem., NonWordRem., LemmatizeRep.	0.751
86	Downcasing, StopWordRem., NonWordRem., NamedEntityRem.	0.746
87	Downcasing, StopWordRem., NonWordRem., PositionAnno.	0.732
88	Downcasing, StopWordRem., NonWordRem., POSRep.	0.699
89	Downcasing, NonWordRem., POSRep.	0.671
90	POSRep., NonWordRem.	0.660
91	Downcasing, POSRep.	0.659
92	POSRep.	0.653
93	POSRep., StopWordRem.	0.636

Table 4.7: Best and worst performing preprocessing pipelines for bigrams and the SVM classifier. The ranking is determined by ordering the pipelines with respect to the achieved F-score.

The resulting bigrams are: *my man, man and, and me, me like, like in.* There are now just five bigrams which are much more common and the knowledge gained from using this tweet for training is probably much more transferable to other tweets due to the lack of specific entities which probably will not even occur in any other tweets. Hence, the learner seems to be able to generalize much better because there is much less noise due to entities and it has more information available. One may argue that intuitively it is much more desirable to have the system learn a structure where the entities are replaced and not removed, which would result in the following text for the example tweet from above:

"my man ||USER|| and me like ||LOCATION|| in ||LOCATION|| ||URL||"

While this approach indeed seems to make more sense intuitively, the data suggests it is slightly sub par to simply removing the entities. However, the differences are not statistically significant and hence the problem becomes more of a design decision than a necessity for good performance.

As it was expected, the lowest ranks are taken by pipelines involving POS replacement. However, the absolute performance of pipelines using only bigrams of POS tags as features is still higher than the majority vote baseline. Therefore, these classifiers may still be useful for some use cases. Another tendency which could already be observed for

unigrams can be confirmed for bigrams as well: Removal of stop words and non-words seems to have consistent negative effects on performance compared to the baseline of doing no preprocessing at all. The only difference between rank 86 and rank 1 is the addition of stop word removal and non-word removal for the rank 86 pipeline which results in an absolute decrease in F-score of about 0.1. Tokens commonly considered to be noise seem to be useful indeed, especially for higher order n-grams where the absolute performance drop relatively to the top pipelines is greater than for unigrams.

Trigrams

In table 4.8 the results for trigrams are presented. The main difference to note when comparing the top pipelines with the lower order n-grams is the dominance of preprocessings coming from linguistics. For trigrams it seems that preprocessing techniques such as POS annotation have greater potential than for unigrams or bigrams. One possible explanation may be that trigrams are able to capture longer sentiment bearing phrases, and having the part of speech tags of those phrases at hand may alter the sentiment some phrases carry. A phrase in past tense may express regrets while the same phrase in present tense may express a positive sentiment such as joy. For example, consider "I liked X" vs. "I like X". The first expression implies that the author does not like "X" anymore and does not carry the same sentiment as the second expression in present tense, where the other still likes "X".

However, the pipeline ranked first qualifies this hypothesis. The top pipeline consists of downcasing and lemmatization. While it seems beneficial to annotate tokens with their POS tags, it seems to be even better to perform more drastic normalization such as reducing each token to its lemma. Part-of-speech information is completely lost when performing lemmatization only. The beneficial effects of mapping all forms of a word to its lemma and the resulting increase in information about the sentiment bearing of the token seems to outweigh the benefits of making use of the POS tags. Both methods are certainly viable and may be sensitive to different kinds of sentiment bearing phrases and can be made use of for ensemble building or other methods requiring diversity in the classifiers involved.

The tendencies for the lower ranked pipelines observed for unigrams and bigrams can also be noted for trigrams and further strengthen the hypothesis that the removal of stop words and non-words is consistently decreasing performance.

Subgrams

Table 4.9 consists of the best and worst pipeline configurations for subgrams of size 3. The data for subgrams of size 2 and 4 is not presented here, since the results are qualitatively very similar to those of size 3.

Subgrams of all sizes seem to benefit the most from preprocessing steps which are drastically decreasing the number of tokens in the data. All the top ranking pipelines contain removal or replacement of user names, URLs and named entities. The explanation for this behavior is probably the same as for higher order n-grams. Unique tokens such as

Rank	Preprocessing Pipeline	F_1
1	Downcasing, Lemmatization	0.818
2	Downcasing, POSAnno.	0.813
3	Downcasing, AtMentionRepl.	0.811
4	Lemmatization	0.810
5	Downcasing	0.808
26	-	0.792
92	Downcasing, StopWordRem., NonWordRem., NegationAnnot.	0.500
93	Downcasing, StopWordRem., NonWordRem., NamedEntityRem.	0.492
94	Downcasing, StopWordRem., NonWordRem., PositionAnnot.	0.435

Table 4.8: Best and worst performing preprocessing pipelines for trigrams and the SVM classifier. The ranking is determined by ordering the pipelines with respect to the achieved F-score.

user names, URLs and named entities add a lot of character combinations which are unseen in normal language. The huge number of features that are actually just noise make up a large percentage of the entirety of subgram features. Hence, it makes sense that significantly reducing the noise leads to increased performance.

Another interesting observation can be made when looking at the pipeline at rank four. This pipeline is the only top ranking example so far which includes the removal of stop words. At first sight stop word removal seems to be beneficial for subgrams but when taking a closer look one notices that the pipeline at rank four only differs from the top ranking pipeline in regard of stop word removal. Thus, stop word removal is probably not as harmful as for other features but still slightly degrades the performance of well performing pipelines. Moreover, stop word removal and non-word removal are still present in the lower ranking pipelines.

Precision and Recall per Feature Extraction Method

Figure 4.2 provides some deeper insights by presenting precision and recall for the positive and negative class by feature extraction method. While the box plots for positive precision and negative recall look qualitatively very similar to the one presented in 4.1, the remaining two are fairly different. Most notably, negative precision outlines an interesting effect. The performance ranking of the n-grams is inverted for negative precision compared to the overall F-score. Trigrams have the highest average, mean and absolute value with much less variance than for the F-score. One possible expq:lanation for this might be the ability of trigrams to capture distinctive positive phrases as a whole. Note, that this does not seem to be the case for negative phrases where trigrams still are outperformed by bigrams. This might be an indication that positive phrases tend to be longer than negative phrases. Furthermore, the NBC consistently shows higher negative precision than the SVM. The effects on subgrams and skigprams are qualitatively and relatively very similar to the behaviour of n-grams, shown previously in figure 4.1.

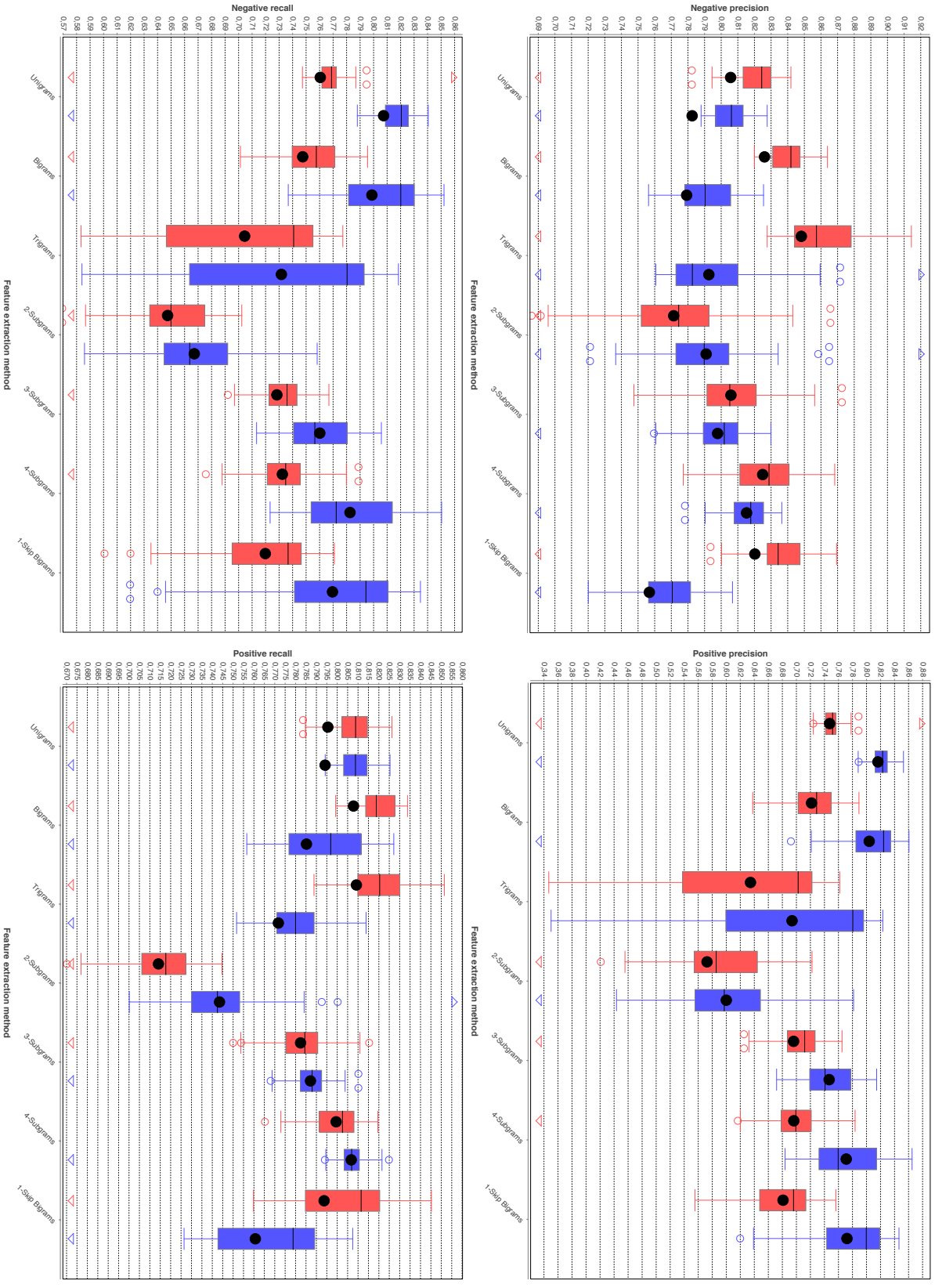


Figure 4.2: Comparing positive/negative precision/recall for the feature extraction methods. Chart type is the same as in figure 4.1.

Rank	Preprocessing Pipeline	F_1
1	AtMentionRem., URLRem., NamedEntityRem.	0.806
2	AtMentionRep., URLRep., Downcasing, NamedEntityRep.	0.803
3	AtMentionRep., URLRep., NamedEntityRep.	0.802
4	AtMentionRem., URLRem., StopWordRem., NamedEntityRem.	0.798
5	AtMentionRep., NonWordRem., URLRep., Downcasing, NamedEntityRep.	0.795
35	-	0.759
74	StopWordRem., NonWordRem.	0.713
75	NamedEntityRem., Downcasing	0.711
76	NamedEntityRem.,	0.704
77	NamedEntityRem., StopWordRem.	0.700
78	StopWordRem., NamedEntityRep.	0.700

Table 4.9: Best and worst performing preprocessing pipelines for subgrams-3 and the SVM classifier. The ranking is determined by ordering the pipelines with respect to the achieved F-score.

4.3 Overview Comparison of Preprocessing Techniques

While the previous section is dedicated to assessing the sensitivity of feature extraction methods to the choice of preprocessing pipeline, this section is focussed on comparing the overall performance of the investigated preprocessing techniques.

Figure 4.3 presents the overall performance per feature extraction technique. The evaluation methodology is the same as in the previous section. Only a single preprocessing technique is used at a time. Each box consists of five values, one for each feature extraction method. The baseline consists of no preprocessing method at all.

First of all, the NBC is consistently outperformed by SVM, like it was the case in the results presented in the previous section 4.2.

Moreover, *PartOfSpeechAnnotation* has the best average and mean performance by a relatively large margin for SVM while it has to share its standing with *AtMentionRemoval* for NBC. While *PartOfSpeechAnnotation* weakens the performance of some feature extraction methods, its high average performance hints at a lot of potential for positive effects.

The overall maximum performance is achieved by *NegationAnnotation* for SVMs. Explicit handling of negation - even in a very basic and simple way as it was done for this experiment - obviously yields strong benefits. An explanation for its effectiveness lies in the fact that sentiment bearing words and phrases may express the total opposite of their sentiment when negated.

Another effect of note is the negative influence of *StopWordRemoval*, it seems to consistently perform worse than the baseline in most cases. While it makes sense intuitively to remove words which apparently carry no useful information in general this does not seem to be the case for sentiment analysis of informal short texts such as tweets.

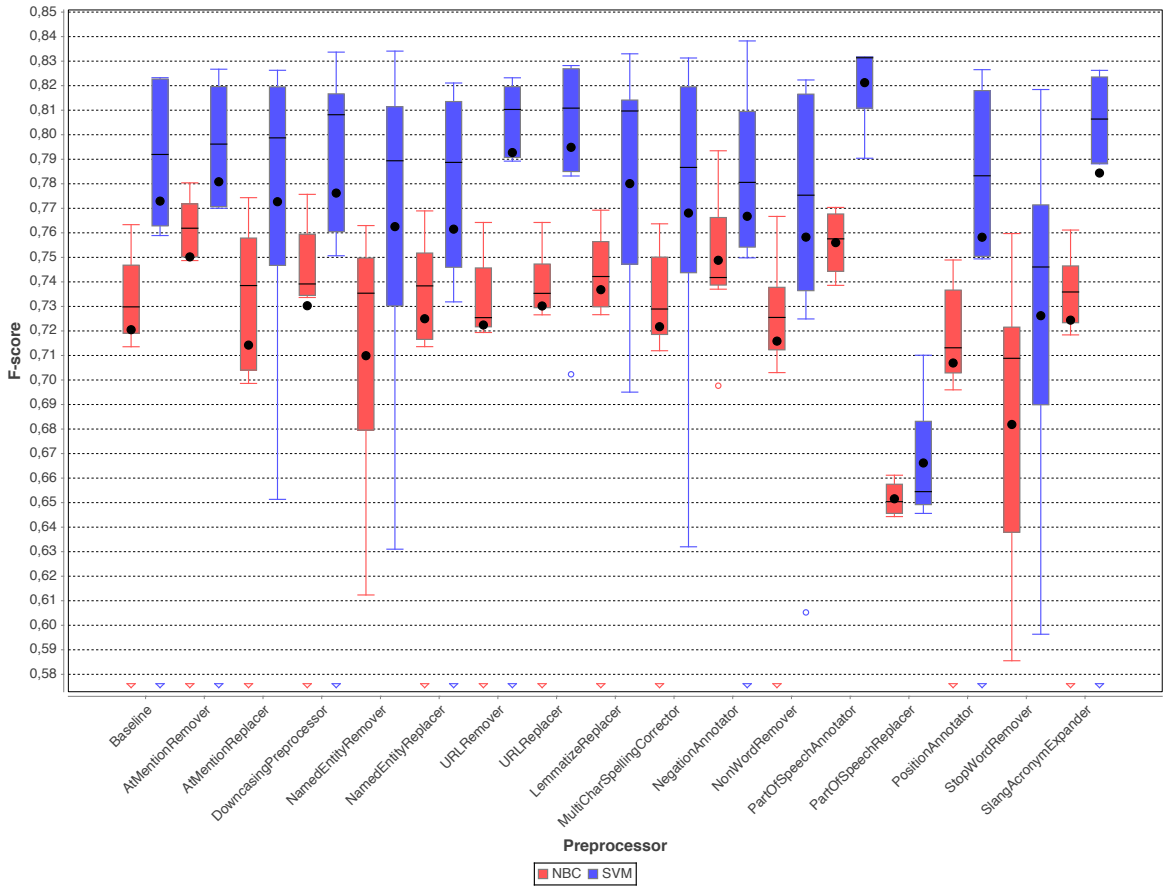


Figure 4.3: Overview comparison of preprocessing techniques. The black dots denote the mean, the black line the median and the unfilled circles and triangles denote outliers.

Replacement of named entities and URLs improves performance in some cases and mostly has no negative effects. A possible explanation for this effect may be the drastic reduction of noise when removing entities and URLs. However, removal of the named entities yields significant drops in performance sometimes.

For user mentions, the effect is inverted, replacing them with dummy tokens has large variance while completely removing them yields notable benefits. Most of the time a user mention is not really a part of the actual message but only used to notify a friend or follower of the message by gaining his attention through mention of his user name, for example:

> "I'm really sad today. Please help me. @someFriendsUserName"

As was expected, *PartOfSpeechReplacement* - replacing each token with it's POS tag - performs significantly worse than the baseline but still much better than a majority vote baseline. This is a strong indication that the POS tags alone carry some useful information for sentiment analysis and should be considered as useful features.

4.4 Quality Criteria for Preprocessing Techniques

The previous sections presented a thorough investigation of preprocessing techniques and their combinations. However, it is still an open question why some preprocessing techniques work better than others. For this reason, the possibilities of deriving general quality criteria for preprocessing techniques from the results presented previously are discussed in this section.

First, it was investigated if there is a correlation between the size of the feature space - i.e. the number of distinct tokens, called *corpus size* in the following - and the resulting performance. Since most of the preprocessing techniques used for Twitter sentiment analysis aim at reducing noise, the aforementioned correlation is a reasonable assumption because the reduction of noise - the removal of tokens - sometimes drastically reduces the corpus size.

Looking at the scatter plot presented in figure 4.4 it becomes clear that this simple assumption does not hold. While the qualitative feel, when looking at the plot may be that the performance drops with increasing corpus size, a notable correlation could not be found.

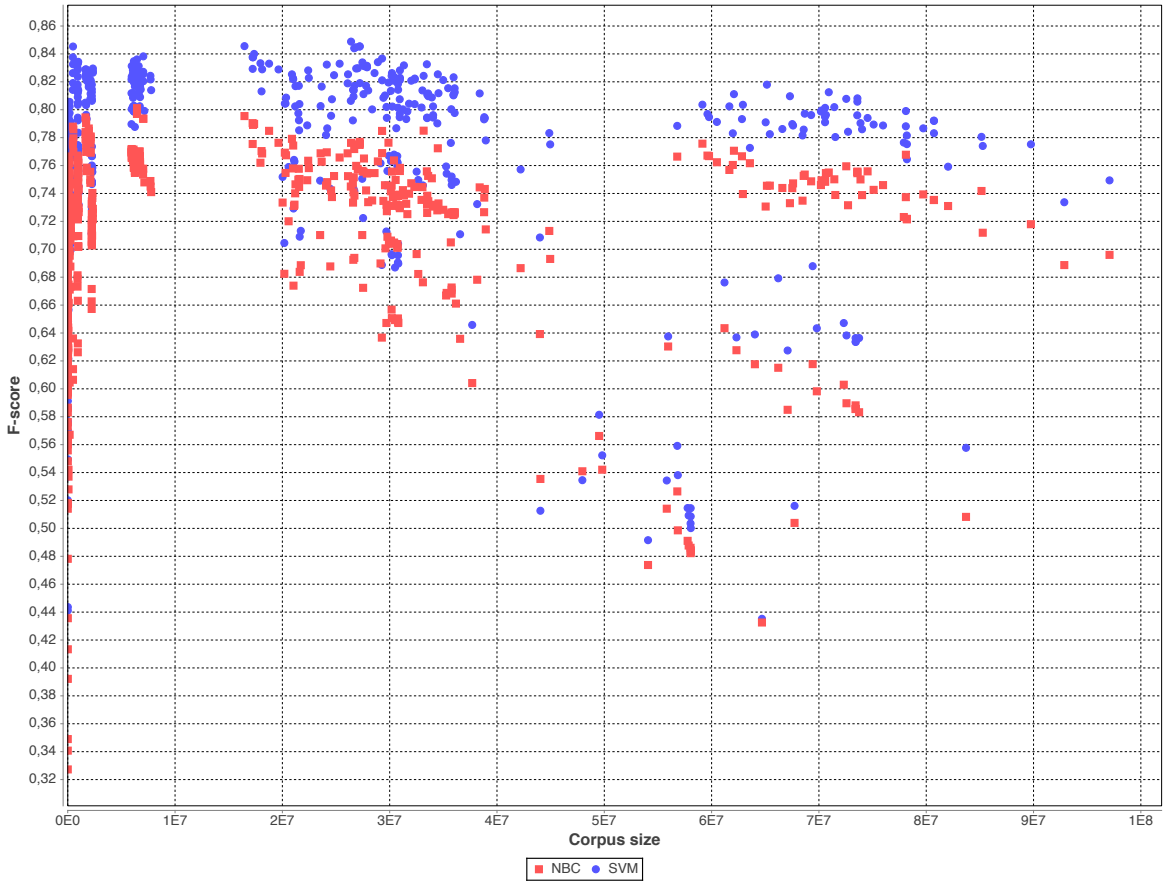


Figure 4.4: Scatter plot of the corpus sizes of all investigated pipelines and their respective macro F-score.

However, the plot in figure 4.4 does not illustrate other parameters of the pipelines

but the size of the corpus. Higher order n-grams for example tend to induce a much higher corpus size with increasing n (see table 4.2) while showing declining classification performance. When relating the corpus sizes in table 4.2 of the feature extraction methods to the results shown in figure 4.1, one has to conclude that there is no direct connection between absolute corpus size and performance. Pipelines using trigrams as feature extractor consist of double the number of tokens compared to bigrams but their performance is usually worse than that of bigrams. Therefore, absolute corpus size does not seem to be a good quality criterion for preprocessing techniques.

Next, the relative corpus size was investigated. For this work the relative corpus size *rcs* of a combination of preprocessing technique and feature extraction method is defined as:

$$rcs = \frac{\text{corpus size of the feature extraction method with preprocessing}}{\text{corpus size of the feature extraction method without preprocessing}} \tag{4.1}$$

The results for the investigated pipelines are shown in figure 4.5. Unfortunately, no correlation between F-score and relative corpus size could be found. Relative corpus size is also ruled out as an indicator for well performing preprocessing techniques.

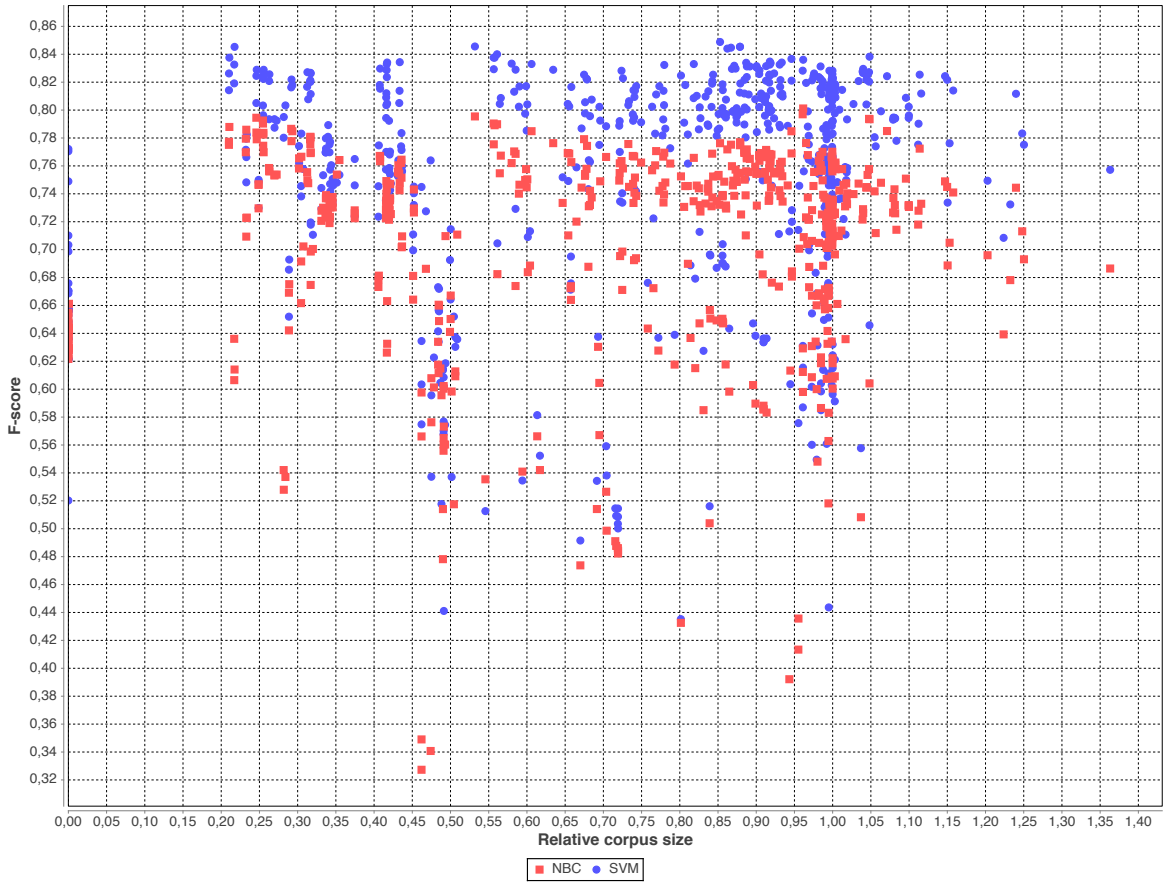


Figure 4.5: Scatter plot of the relative corpus size for all investigated pipelines with their respective macro F-score.

Simply shrinking the corpus size by reducing the number of distinct tokens used for the classification has - in general - no measurable positive effect. Actually, most of the used techniques are not aimed towards just reducing the corpus size but rather normalizing the data. The overall goal is to condense information in a way that allows the machine learning systems used for classification to make better use of the data. Consider for example the following tweet:

> "I looooooooooooooove the new iPhone."

Another user may also express positive sentiment towards the new iPhone, also using the word "love" such as this:

> "I really looove the iPhone 6."

Both authors made use of the positively connoted word "love" to express that they like a product. Nevertheless, they both used a variable number of the letter "o" in their tweets. To a human it may be obvious that they both are saying the same thing, to a machine the tokens are totally independent tokens. Normalizing all occurrences of "love" with more than one letter "o" to "loove" allows the algorithm to recognize both expressions as the same token. While this normalization reduces the corpus size, it also increases the amount of knowledge a learner can gain from the training data.

These thoughts lead to the concept of the *coverage ratio*. Let F_{test} be the set of distinct features extracted from the test dataset by a pipeline and F_{train} be the set of distinct features extracted from the training dataset using the same pipeline as for F_{test}, then the coverage ratio c is defined as

$$c = \frac{|F_{\text{train}} \cap F_{\text{test}}|}{|F_{\text{test}}|}. \tag{4.2}$$

The coverage ratio can be seen as a measure that expresses how well the tokens in a test dataset are covered with a given training dataset. It is expected that preprocessing techniques that increase the coverage ratio also should have positive influence on the classification performance. The results presented in figure 4.6 strengthen the aforementioned hypothesis.

Pipeline configurations with higher coverage ratios tend to yield higher F-scores. However, the hypothesis breaks when $c \approx 1$. Pipelines with such a high coverage ratio usually map many tokens to the same target token, for example pipelines including *PartOfSpeechReplacement*, where all tokens are replaced with their respective part-of-speech tag. Since there are usually only a few dozen distinct POS tags, the coverage ratio is 100%. Ignoring the outliers generated by these effects the correlation coefficients are $r = 0.87$ for the NBC and $r = 0.88$ for the SVM classifier.

A high coverage ratio seems to be a reasonable quality criterion for assessing the value of a preprocessing technique or a whole pipeline of preprocessing and feature extraction. However, one has to be cautious when $c \approx 1$, since preprocessing techniques mapping many tokens to the same token may still not be very useful despite their high coverage ratio. Moreover, one should keep in mind that this measure should only be used for

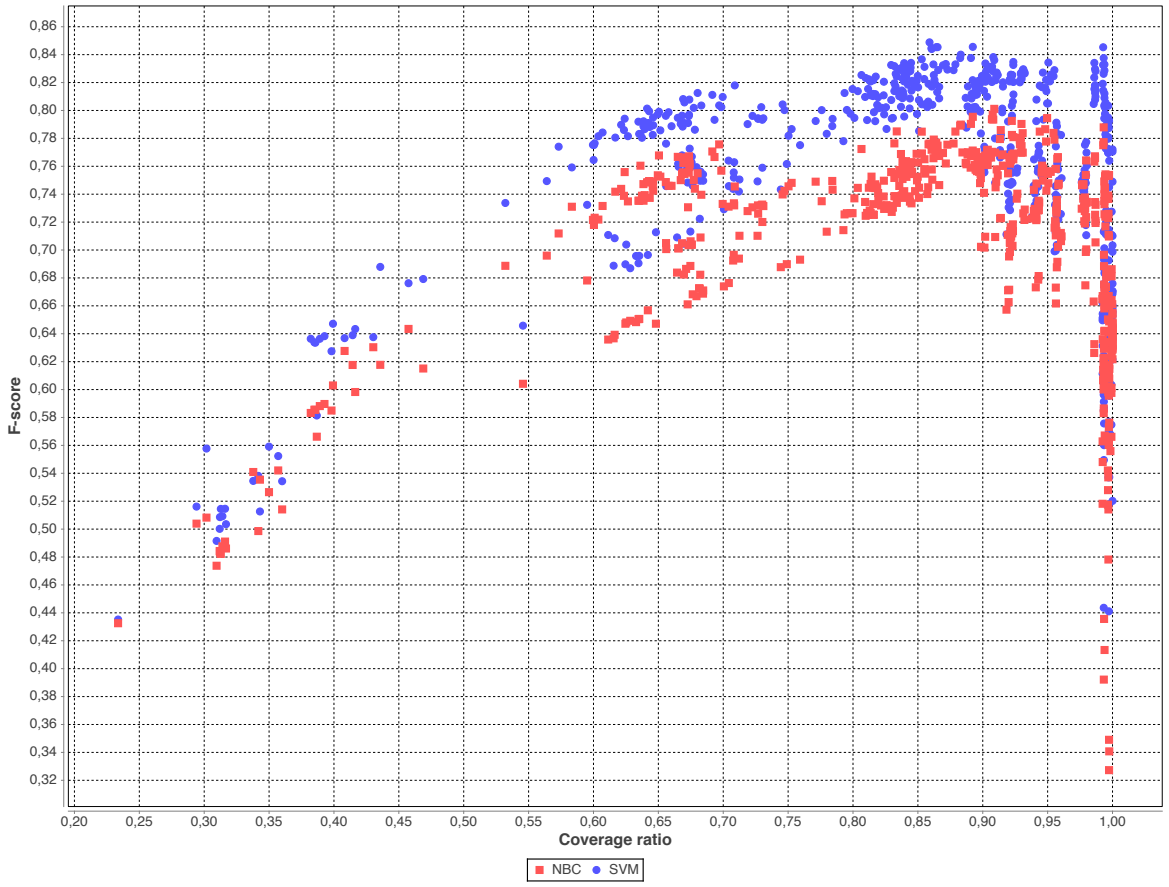


Figure 4.6: Scatter plot of all investigated pipelines and their respective macro F-score.

preprocessings aiming at increased normalization of the data. Annotating negations for example does usually not increase the coverage ratio but yields the best performance improvements in some cases.

4.5 Summary and Conclusion

Previous results from from Neubauer (2014) have indicated that most of the investigated preprocessing techniques have no significant effect on the classification performance. However, the study did not investigate all preprocessing techniques used in the current state of the art methods and also did not investigate any combinations of them. Moreover, the study only focussed on unigrams as feature extraction methods.

To shed some light on the open questions, a more in-depth study was performed and investigated more than 700 combinations of preprocessing techniques, feature extraction methods and classifiers. The results indicated that different feature extraction methods are affected very differently by the various preprocessing techniques. Especially interesting is the insight that n-grams of different orders lead to diverse results when used with the same preprocessing techniques.

Next, the preprocessing techniques were compared in greater detail. As it was expected,

their effects vary in performance. It could be shown that - contrary to popular believe - the removal of stop words diminishes performance in most of the investigated cases and should be used with caution. Other preprocessing techniques such as replacing words with their POS tags perform significantly worse than the baseline but still perform a lot better than guessing or a majority vote strategy. Hence, they might provide useful diversity for building of ensembles.

The results of the final investigation indicated that the *coverage ratio* of a pipeline can be a useful criterion for estimating the usefulness of preprocessing techniques that aim at normalization of the text data.

Finally, it can be noted that SVM classifiers outperformed Naive Bayes classifiers in almost any investigated case which is in line with the results of other researchers[1]. Thus, Naive Bayes classifiers will no longer be considered for the remainder of this thesis.

Combining all the aforementioned insights, a recommendation (see table 4.10) can be given as to which preprocessing techniques to use for certain feature extraction methods, when building bag-of-word base models for Twitter sentiment analysis.

Feature Extraction	**Preprocessing**
All	NamedEntityReplacer, URLReplacer, AtMentionReplacer, MultiCharSpellingCorrector, Downcaser
Unigrams	NegationAnnotator
Bigrams	-
Trigrams	LemmatizeReplacer
Subgrams{2,3,4}	-

Table 4.10: Recommended preprocessing techniques by feature extraction method.

Some of the suggested preprocessing techniques have no significant positive effect on the classification performance but they often reduce the dimension of the data without having a negative impact on performance. As more of a design choice, their use is still recommended, since they yield a small benefit without a significant drawback.

[1] See for example Firmino Alves et al. (2014), Neubauer (2014).

5 Reliable General Purpose Twitter Sentiment Analysis

Twitter sentiment analysis usually denotes the task of assigning a given tweet a sentiment label of either *positive* or *negative* and is an integral part of many practical applications[1]. Some researchers consider *neutral* as a third class. However, defining a neutral class is a hard task. Pak and Paroubek (2010) for example label tweets of popular news sites as neutral. This assumption is not always true. Another common approach is to label tweets as *neutral* which contain no sentiment bearing phrases at all (Nakov et al. 2013, Rosenthal et al. 2014; 2015). But what should be done with messages containing inconsistent sentiments? The world is not always black and white like it is often proposed. For some messages a distinct sentiment cannot be chosen but neither are they truly *neutral*. This becomes a problem when one desires to analyze the unfiltered public stream of tweets since it will contain a large portion of these tweets. Examples for applications where this is relevant are real-time event and topic detection systems or systems aiming at determining the sentiment of a single person, group of people or geographical regions such as countries.

After more formally defining the problem in the following section 5.1, the creation of a high quality dataset that is representative for the public Twitter stream, is described in section 5.2. Next, there are multiple sections which provide experiments and results for a systematic evaluation of a representative subset of archetypical state-of-the-art methods from a related domain and how well they can be transferred to general purpose twitter sentiment analysis. Methods based upon manual feature engineering are discussed in section 5.3. Section 5.4 consists of a report for the feasibility of the basic deep convolutional neural network architecture for text classification, while section 5.5 provides insights about efficiently making use of active learning in conjunction with deep neural networks. Following that neural network investigation, section 5.6 presents a solution to the spam classification problem. Finally, the results are summarized in section 5.7.

5.1 Problem Definition

An alternative approach to the labeling of general tweets is proposed in the following. Its basic idea is the explicit incorporation of sentiment uncertainty. Many applications only make use of the sentiment bearing tweets. Hence, dividing the messages into positive/negative and everything else usually is sufficient. It is often not that important whether a tweet is truly neutral or if its sentiment could just not be distinctly chosen with enough confidence. Moreover, for the use case of analyzing the unfiltered public

[1] See chapter 1 for an overview.

stream of tweets it is more important to be sure of the analysis results than to base the results upon more but possibly inaccurate tweets. Thus, tweets with contradicting sentiments should not be mislabeled as positive or negative in most cases. Due to the large number of tweets generated every day it is practical to discard messages with lesser confidence and focus on those where a true sentiment can be detected. They do not have to be finally discarded, one can still run advanced analysis methods to try and extract useful sentiment bearing information. The effort may not be worth the outcome though, because of the huge amounts of data which is available, one can be generous in discarding messages. Hence, a more detailed analysis is not considered to be the main scope of this work and is left for future research in the field.

When analysing the public stream of tweets, we are interested in use cases where the data is interpreted as "Electronic Word of Mouth" (Jansen et al. 2009), i.e. the personal messages of private Twitter users. Hence, we would like to filter out automatically generated messages commonly produced by mobile applications, bot messages and obvious advertisement. These unwanted messages are called *spam* throughout the rest of this thesis. *Spam* adds unwanted noise by polluting topics with artificially injected tweets which dilutes the desired representation of personal opinions. Some schematic examples for spam messages are shown below:

> "I have X new items of kind Y in Game Z!!!! Join me ||URL||."
> "Make $XXXX dollars per day by filling out awesome surveys! Start now: ||URL||"
> "Buy the new product X. It is so good! Get a discount JUST TODAY!!!! ||URL||"

Many publications dealing with spam detection on Twitter (Grier et al. 2010, Lin and Huang 2013, Lee et al. 2011, Song et al. 2011, Mccord and Chuah 2011) use a different definition of spam, mostly in the sense of "Spam messages are messages containing harmful links or are otherwise harmful for the users.". In addition, the focus lies on catching the users generating the spam by looking at the accounts' behaviour over time. When performing real-time analysis, a given tweet has to be determined to be spam or no spam by looking at its content and meta data only, as there is no time to examine the author's account in detail. Therefore, most of the published methods are not applicable for the scenario at hand due to their different definition of spam or their lack of real-time capabilities and new strategies must be explored for the spam definition needed for the applications of interest.

Summing up all of the above, the task of *reliable general purpose Twitter sentiment analysis* can be defined as follows:

For a given tweet $t \in T$ where T is the set of all tweets find a function $RGPTSA : T \rightarrow$

{positive, negative, uncertain, spam} so that

$$RGPTSA(t) = \begin{cases} \text{positive} & \text{when } t \text{ is no spam and the sentiment is distinctly positive} \\ \text{negative} & \text{when } t \text{ is no spam and the sentiment is distinctly negative} \\ \text{uncertain} & \text{when } t \text{ is no spam and the sentiment cannot be distinctly determined} \\ \text{spam} & \text{when } t \text{ is obvious content pollution such as advertisement or bot generated messages, regardless of the sentiment} \end{cases} \tag{5.1}$$

Note, that tweets usually labeled as *neutral* can be assigned to the class *uncertain* too, as they provide no additional information for the sentiment analysis and can be treated in the same way as tweets of *uncertain* sentiment. This approach reduces the noise for the sentiment bearing classes which is a desirable feature if political or business decisions are supposed to be supported by the analysis results.

5.2 A High Quality General Purpose Dataset

This section is an extended version of already published results (Haldenwang and Vornberger 2015). After introducing the relevant publicly available datasets and discussing why they are not appropriate for the underlying use case, a new dataset is introduced and discussed that is suitable for the aforementioned task of reliable general purpose Twitter sentiment analysis.

5.2.1 Available Datasets and Their Shortcomings for General Use Cases

Go et al. (2009) introduced the concept of distant supervision to Twitter sentiment analysis . Tweets are labeled as positive when they contain a positive emoticon such as ":-)" and labeled as negative when they include a negative emoticon such as ":-(". They collected 1.6 million tweets labeled by the occurring emoticons between April 6, 2009 and June 25, 2009. The time period from which the dataset has been sampled is rather short and may be biased towards certain big events which happened in that time frame or even towards the season of the year. Hence, it is not representative for the general public stream. Moreover, 360 tweets have been manually annotated as a test set. There is no information available on how exactly the labeling was done. A test set that small does probably not allow for well-founded results.

Shortly afterwards, another corpus became available whose creators tried to find a way to incorporate the neutral class (Pak and Paroubek 2010). The approach made use of the distant supervision strategy proposed by Go et al. (2009) for the *positive* and *negative* classes. The neutral class is populated with tweets that were collected

from accounts of popular news sites such as the *New York Times*[2]. While the distant supervision approach for the *positive* and *negative* classes seems to work reasonably well and was widely used, the assumption for the neutral class does not fit the common definition of *neutral*. Consider for example the tweet "A dozen of little kids has been killed in the fight.". Most human labelers would assign a negative sentiment to this headline. Even though the incident is reported objectively, the projected sentiment is often not perceived as neutral. Pak and Paroubek (2010) should probably have called their neutral class *objective* to prevent confusion. Moreover, they provide no information in which time frame the tweets have been collected and how exactly the collection was done. In summary, the definition of the class *neutral* seems blurry and since there is no information on how the corpus was collected one cannot know how representative it really is for the unfiltered public stream.

Another group of available datasets are strongly biased towards certain topics such as movie revenue (Asur and Huberman 2010), politics (Diakopoulos and Shamma 2010) or certain special entities such as *Apple, Google, Microsoft, and Twitter* (Liu et al. 2012). These datasets may be useful for their respective domain but are of little use for general purpose sentiment analysis of the public stream.

One of the most popular datasets at the moment which is widely used as state of the art benchmark for Twitter sentiment analysis is provided by the organizers of the annually held *International Workshop on Semantic Evaluation* (Nakov et al. 2013, Rosenthal et al. 2014; 2015, Nakov et al. 2016). They provide a growing number of labeled tweets, since additional data is acquired each year the workshop is held. The data is sampled from a corpus which was collected in the time frame between January 2012 and January 2013 to prevent general topical biases. Popular topics have been extracted through identification of frequently mentioned named entities and only tweets including a mention of those extracted entities are used for the dataset. Furthermore, only tweets scoring above a certain polarity threshold - determined by a sentiment lexicon[3] - were considered to ensure the inclusion of a sentiment. The labels included in the dataset are *positive*, *negative* and *neutral*, determined by a majority vote of five labelers who were told to vote for the sentiment they perceived as strongest, when in doubt. For the purpose of representing the general public stream of tweets the aforementioned filtering process is very detrimental. According to a study presented by Han and Kavuluru (2015), only about 10% of general tweets contain named entities and sentiment bearing words in combination. Due to the biases towards popular named entities and the uncertainty for the labels introduced by the blurred majority vote, the dataset is not representative for the general public Twitter stream and the class definitions do not entirely match those of reliable general purpose Twitter sentiment analysis.

The first - to the best of the author's knowledge - attempt of creating a representative dataset for the public twitter stream was made by Agarwal et al. (2011). They acquired 11,875 manually annotated tweets which were taken from the Twitter sample stream from a commercial source. Since the source is not public, there is no information about the exact sampling and labeling process so it cannot be verified that the data fits the criteria for the use cases investigated in this thesis. Moreover, the dataset also relies

[2] `https://twitter.com/nytimes`, last checked at 01.08.2016.
[3] The lexicon used was *SentiWordNet*, see Baccianella et al. (2010) for details.

on the classic definition of *neutral* which is not in line with goals of this work. Han and Kavuluru (2015) labeled additional 1,000 general tweets and combined them with the dataset of Agarwal et al. (2011). Their results indicated that methods performing well for datasets which are biased towards named entities in conjunction with sentiment bearing words are not necessarily performing as well for general tweets. However, the used dataset also does not fit the exact criteria needed for the underlying use cases of this thesis due to the differing definition of *neutral* and the lack of a *spam* label.

In summary, it can be stated that there is - to the best of the author's knowledge at the time - no publicly available dataset which fits the criteria of the use case at hand. Hence, a new dataset[4] needed to be created which is detailed in the following section.

5.2.2 Creating A General Purpose Dataset

To acquire a representative view on the label composition of the public Twitter stream, the data was randomly sampled from a collection of about 33 million English tweets[5] with their creation dates ranging from June 2012 to August 2013 to minimize topical bias. Each tweet was labeled by two human labelers who had to assign it one of the labels *positive*, *negative*, *uncertain* or *spam*. Having two labelers annotate each tweet reduces the effects of subjectivity which play an important role in sentiment analysis, since sentiment is always kind of subjective. In total, 14,506 tweets have been labeled by 27 labelers. The labelers consisted of master students and researchers from the *Media Computer Science Group*[6] of the *Institute of Computer Science*[7] of the *University of Osnabrück*[8], Germany.

The labeling process was carried out in a custom web application[9] of which a screenshot is shown in figure 5.1. The tweet to be labeled is presented in the center along with four buttons for each of the classes. Right beside the labeling interface, the instructions on how to annotate tweets are shown to the user. The instructions can be hidden to reduce the distraction. Furthermore, when a user accidentally pressed a wrong button, he can always go back and correct the mistake. Overall, the annotation procedure is designed to be as stress free and as little distracting for the annotators, as is possible.

The distribution of labels is shown in figure 5.2. There is a total of 9,356 tweets (64.5% of total tweets labeled) to which both human labelers assigned the same label. The annotators assigned the label *spam* 15% to and 55% were labeled *uncertain*. A definite sentiment label could only be assigned to 30% of tweets with 13% being positive and 17% being negative.

[4] The dataset is publicly available at `http://project2.informatik.uos.de/TweeDOS`.

[5] The collection was first introduced in Neubauer (2014).

[6] `http://www.informatik.uni-osnabrueck.de/arbeitsgruppen/medieninformatik.html`, last checked 03.08.2016.

[7] `http://www.informatik.uni-osnabrueck.de/`, last checked on 03.08.2016.

[8] `http://www.uos.de`, last checked on 13.07.2017.

[9] The web application was developed under the author's supervision as part of the bachelor thesis of Miriam Beutel - whome I would like to thank for her excellent work - that is available here: `http://www.informatik.uni-osnabrueck.de/arbeitsgruppen/medieninformatik/abschlussarbeiten/beutel.html`, last checked 03.08.2016.

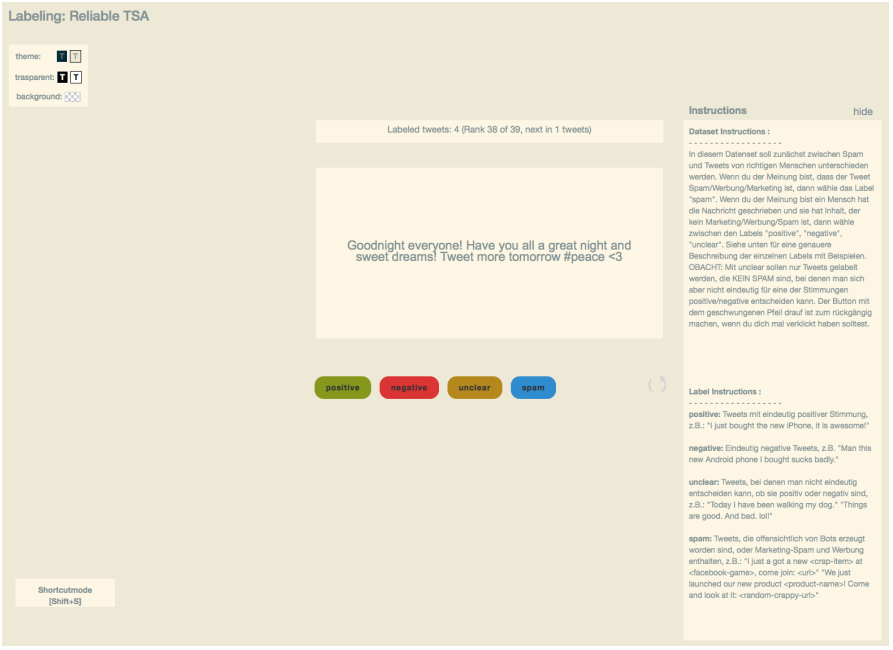


Figure 5.1: Screenshot of the labeling application.

These results provide further evidence for the claim that one has to deal with uncertainty in sentiment analysis when working with the public Twitter stream. First, more than half of the tweets on which the labelers agreed upon are of uncertain sentiment. Second, not even two human annotators are able to agree upon 34.5% of the labeled tweets which affirms the claim that sentiment is subjective.

To formally assess the inter annotator agreement, Fleiss' Kappa Coefficient (Fleiss 1971) was computed resulting in a value of $\kappa = 0.45$ which can be interpreted as *moderate agreement* (Landis and Koch 1977). At first sight this value seems to be rather low but when considering the disagreement matrix shown in table 5.1, the claim of the necessity to deal with uncertainty is further strengthened.

	positive	**negative**	**uncert.**	**spam**
positive	1176	106	1666	143
negative		1620	2263	58
uncertain			5138	914
spam				1422

Table 5.1: Disagreement matrix showing the absolute numbers of label combinations.

Labelers seem to have a very good understanding of what distinguishes the classes *positive* and *negative*, only 106 tweets have been assigned both these labels. The disagreement for *positive/spam* and *negative/spam* is of similar or even smaller magnitude. Looking at these tweets it was noticed that the disagreement is mainly related to misunderstanding of the labeling instructions or probably accidentally clicking the wrong label. Hence, these tweets should be omitted from the test set when evaluating methods for reliable Twitter sentiment analysis.

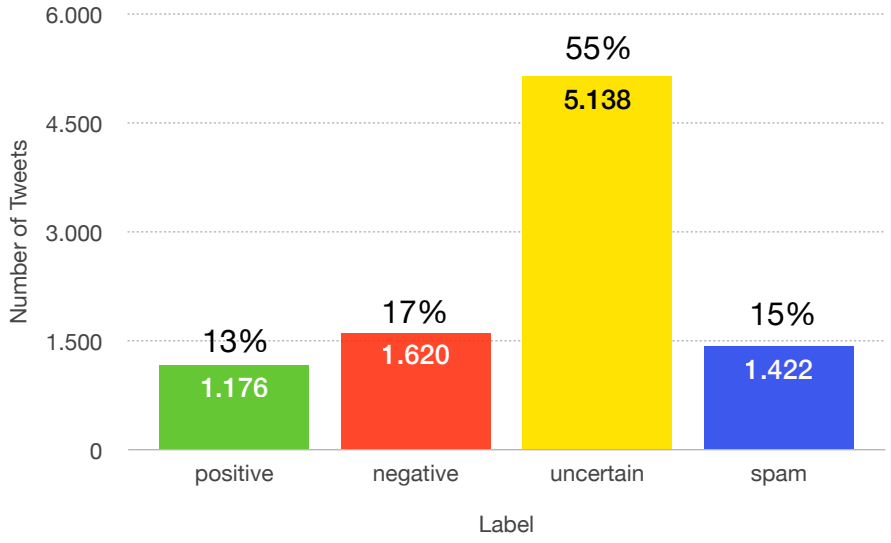


Figure 5.2: Distribution of labels for tweets which both labelers agreed upon.

However, the disagreement between *positive/negative* and *uncertain* is relatively large. These tweets make up about 76% of the tweets to which the two labelers assigned different labels. This indicates that in many cases not even two humans can agree upon whether a tweet contains a distinct sentiment or should be labeled *uncertain*. Systems aiming to perform reliable sentiment analysis of the public Twitter stream should be able to deal with these tweets. While not strictly belonging to the category *uncertain*, they should still be labeled as such or at least not be considered for sentiment analysis. Another possible approach can be to interpret them as *rather positive* or *rather negative*, depending on the amount of reliability the respective application requires.

Moderate disagreement (914 tweets) can be noted for the classes *uncertain* and *spam*. Since these tweets may still contain useful information in the sense of answering the question "What do people talk about?", they probably should not be considered spam. Nevertheless, they neither should be assigned a sentiment. A system labeling these as *uncertain* will still produce reliable results with regard to sentiment analysis.

As a first approach one can make use of just the tweets with two identical labels to asses methods for reliable sentiment analysis of the public Twitter stream. However, it should be considered that in practice the tweets upon which the labelers disagreed can also appear in the stream and have to be handled to provide reliable sentiment results. To enable researchers to develop systems which meet all the aforementioned requirements the complete dataset including the tweets disagreed upon is publicly available.[10]

5.2.3 Summary and Conclusion

When performing analysis on the public live stream of Twitter with regard to sentiment and opinions of people, it needs to be considered that more than half of the tweets cannot

[10]The dataset is available at `http://project2.informatik.uos.de/TweeDOS`, last checked on 18.07.2017.

be assigned a distinct sentiment with the labels *positive* or *negative*. These tweets have to be filtered or explicitly dealt with before sentiment analysis takes place; they are labeled as *uncertain*.

Moreover, one has to deal with *spam* tweets in the sense of automatically generated content pollution. *Spam* adds unwanted noise by polluting topics with artificially injected tweets. Most of the work on spam detection on Twitter does not only use a different definition of *spam* but focusses on catching the users generating the *spam* by looking at the accounts' behaviour over time. When performing real-time analysis, a given tweet has to be determined to be *spam* or no *spam* by looking at its content and meta data only as there is no time to examine the author's account in detail.

New methods have to be explored, which are able to deal with sentiment uncertainty and *spam*, if reliable representations of the public opinion are to be acquired from the Twitter stream. The high quality, reasonably large dataset that was created and presented in this section can be used to develop and evaluate methods for reliable general purpose Twitter sentiment analysis.

5.3 Approaches Based on Manual Feature Engineering

The goal of the experiments presented in this section was to assess the viability of combinations of state-of-the-art manual feature engineering and preprocessing techniques with various classification algorithms for general purpose Twitter sentiment analysis.[11] Please see chapter 3 for a in-depth review of the history of the field and the current state-of-the-art.

5.3.1 General Experimental Setup

The tweet data used for the experiments was the following:

related domain This dataset consists of a concatenation of datasets from a related domain - namely SemEval from 2013-2015 (Nakov et al. 2013, Rosenthal et al. 2014; 2015) and the dataset from Neubauer (2014) - where the possible classes were *positive*, *negative* and *neutral*. In total, the dataset consists of about 26,000 tweets.

in-domain test This is the dataset described previously in section 5.2. It is used mostly as a test set to evaluate the classification algorithms for general purpose Twitter sentiment analysis. The dataset consists of about 9,300 labeled tweets.

in-domain dev This dataset was created in the same manner as *in-domain test* but its size is considerably smaller (about 1,000 tweets). There is no overlap between the two datasets. It is mostly used as development set to tune fix parameters of the classification algorithms with grid search.

[11] See section 2 for an overview of the relevant methods.

The feature extraction methods investigated (called *feature groups* in the following) are a combination of the archetypical methods used in state-of-the-art approaches[12]. They will be referred to as:

uni,bi,tri,quad One of the most basic features are *n-grams* as described in section 2.1.1. Investigated was $n \in \{1, 2, 3, 4\}$.

w2v Word embeddings created with *word2vec* as described in section 2.1.2.

cmu Word cluster features as described in section 2.1.1.

sub2, sub3, sub4 Sub word-level n-grams as described in section 2.1.1 with $n \in \{2, 3, 4\}$.

lex-nrc, lex-mpqa, lex-tweetos, lex-opinion Lexicon features as described in section 2.1.1. The lexicon *lex-tweetos* was generated within the course of this project, in the same manner as *lex-nrc* but using the distantly supervised emoticon corpus from Neubauer (2014).

Preprocessing techniques as introduced in section 2.1.1 are applied to the respective feature extraction techniques according to the results presented in section 4.

Each feature group is first investigated in isolation, meaning the feature vectors are made up only of the data generated by the respective feature group. The idea here is to assess the general feasibility of each feature. Next, the feature vectors are created by concatenating the data of all feature groups. Finally, to investigate how much each feature group really adds to the performance, the feature vectors are created by using all feature groups but one.

5.3.2 Choosing Classifiers to Investigate

The most popular classification algorithms for Twitter sentiment analysis[13] are *Support Vector Machines*[14] (SVM) and *Maximum Entropy Classifiers*[15] (MaxEnt). Since those seem to be most promising, they were most closely investigated. *Naive Bayes Classifiers*[16] (NBC) were also popular in the early stages of the field but were more recently consistently outperformed by the more popular alternatives.[17] Hence, it was chosen not to consider NBCs for this experiments. Since there was no data available[18] whether *Decision Trees* are viable for the sentiment analysis of tweets, they were also informally investigated. It became fairly obvious quite soon in the process that decision trees also perform consistently worse than SVM and MaxEnt and hence, they also were not considered for the in-depth investigation presented in this section. Furthermore, it was chosen to concentrate on approaches based on machine learning and not to investigate more linguistically inspired, manually created rule-based systems.

[12] See section 3 for details.
[13] See section 3 for the history of the field and a summary of the state-of-the-art.
[14] See section 2.2.2.
[15] See section 2.2.3.
[16] See section 2.2.1.
[17] See also section 4.5 for further evidence.
[18] To the best of the author's knowledge.

Both classifiers are trained in two different ways. First, the problem is considered as a standard three-class classification problem and the classifier is trained as usual with data for all three-classes. Second, a confidence based rejection version of the classifier is trained only with data from the classes *positive* and *negative* and classifies samples as *uncertain* when the certainty of the two-class classification falls below a threshold. Following in section 5.3.3 is a more in-depth description of the rather novel approach.

5.3.3 Confidence Based Rejection Classification for the Uncertain Class

The concept of *confidence based rejection classification* is founded upon a geometric interpretation of the decision boundary in a linear classifier and the examples used as training data to learn the respective hyperplane[19]. In general, it is a common practice to use the distance (usually L_2 norm) from a data point to the hyperplane dividing the feature space as a measure of confidence for the classification result. Intuitively this hypothesis makes sense since small changes of the hyperplane have stronger effects on examples which are closer to the plane.

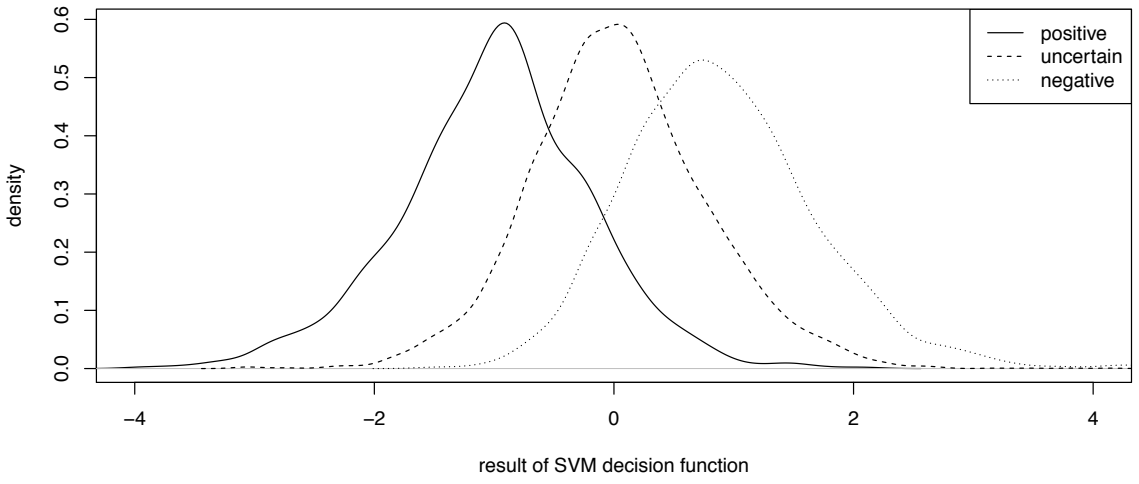


Figure 5.3: Decision function values and their density of a SVM trained with positive and negative tweets for tweets from a general purpose dataset.

Looking at the classes *positive, negative* and *uncertain*, which are needed for reliable sentiment analysis of tweets, a similar structure comes to attention. Conceptionally, *uncertain* examples lie in between *positive* and *negative* ones. Consequently, it can be hypothesized that in a feature space well suited for linearly separating positive and negative tweets, the uncertain tweets are geometrically in between the ones of the other two classes.

[19]The ideas presented in this section were developed by the author in early 2015. However, it should be noted that parallel to the aforementioned efforts, Smailović et al. (2015) published a very similar method based on the related ideas.

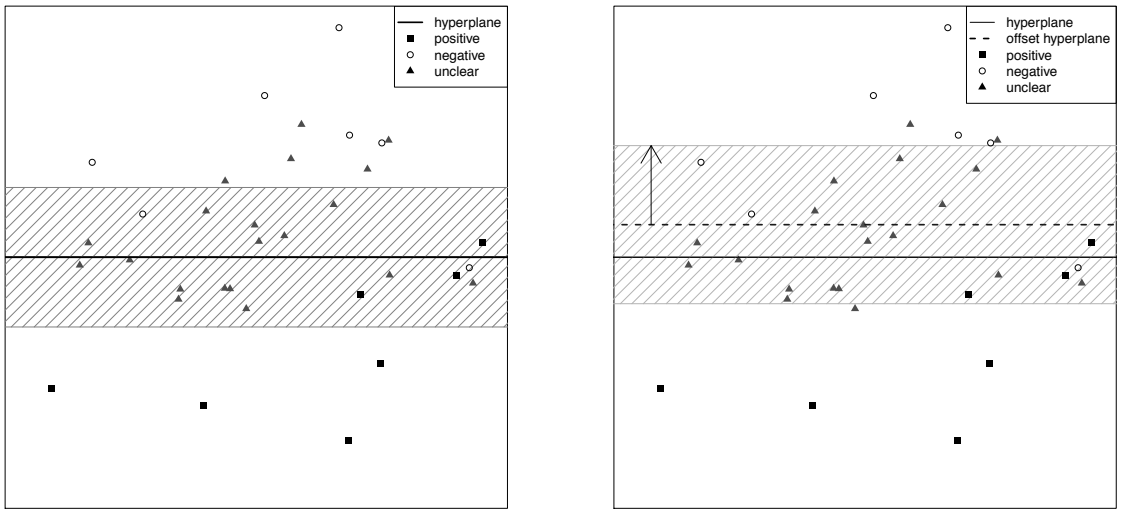


Figure 5.4: Illustration of the confidence SVM idea. The hatched zone denotes the space in which tweets are classified as *uncertain*.

To assess whether the aforementioned hypothesis may be useful for Reliable General Purpose Twitter sentiment analysis, a binary SVM classifier was trained using only the positive and negative tweets from SemEval 2013-2015 (Nakov et al. 2013, Rosenthal et al. 2014; 2015). The feature extractors used were n-grams with $n \in \{1, 2, 3, 4\}$, subgrams with $n \in \{2, 3, 4\}$, word embeddings and lexicon features with the respective preprocessing pipelines according to the results presented in section 4.5. After training the binary classifier the value of the decision function (signed distance to the hyperplane) of all tweets in the dataset presented in section 5.2.2 are computed. Figure 5.3 shows a density plot of the decision function results grouped by the three classes *positive, negative* and *uncertain*. The results strengthen the hypothesis that tweets of uncertain sentiment lie in between positive and negative tweets. Tweets of the class *uncertain* are distributed directly around the hyperplane while tweets of the classes *positive* and *negative* are farther away in the respective directions.

A classifier solving the three-class problem can be created by introducing a zone around the hyperplane. Tweets within the zone are classified as *uncertain* and tweets outside of the zone are classified with the respective class as usual. Figure 5.4a illustrates the aforementioned idea. After noticing that the hyperplane is often not the exact center of the class distributions, an offset parameter for the uncertainty zone is added which simply moves the zone orthogonally to the hyperplane.

Training the confidence based rejection classifier is done by training a binary classifier with *positive* and *negative* data and then optimizing the uncertainty zones' threshold and offset via grid search on the development dataset *in-domain dev* consisting of *positive*, *negative* and *uncertain* tweets.

The approach can also be transferred to other classification algorithms, as long as one can generate some kind of reasonable confidence score which works similar to the

distance used in the aforementioned remarks for linear decision boundaries[20].

5.3.4 Transfer Learning Approach

The main purpose of the transfer learning experiment was to investigate how well a relatively large amount of data from a closely related domain can be used for training a classifier to solve the general purpose sentiment analysis task. The main differences between the in-domain data and the related data are that the related data mostly is not representative for the public Twitter stream and that the classes are not equivalent to those of general purpose Twitter sentiment analysis[21]. However, the class *neutral* is used as training data for the class *uncertain* to evaluate how closely related the classes really are. Moreover, the data was in part collected at a different time frame which may pose a problem because of topical variations and short-dated vocabulary shifts that can occur in informal language such as that often used in tweets.

Training was performed using the *related domain* dataset and the reported macro F_1-scores were acquired by evaluating the trained classifiers on the *in-domain test* dataset.

The results for the single feature group configuration are shown in table 5.2. Subgrams with $n = 4$ being the best standalone feature group (best feature group for confSVM, SVM and confMaxEnt) was quite unexpected beforehand since the experiments conducted previously[22] indicated n-grams with $n = 2$ to be the most promising standalone features. This indicates that words or respectively parts of words with a length of four often transport strong sentiment information. Some sentiment bearing words of this length that first come to mind are for example "good, bad, sad, nice".

Moreover, it can be noted that the confidence based rejection variants of both classifiers outperform the three-class variants in almost all cases. Both confSVM and confMaxEnt perform at the same order of magnitude with little differences for most feature groups. Probably this is the case because the class *uncertain* from the test dataset does not exactly match the *neutral* class from the training data. By training only with *positive* and *negative* samples and assuming anything with lack of confidence to belong to the class *uncertain* this problem seems to diminish.

Many of the investigated feature groups manage to achieve quite a feasible result for a three-class problem with $F_1 > 0.5$.

In figure 5.3 the results of the leave-one-out experiment are presented. As for the single feature groups, the confidence version of both SVM and MaxEnt outperform their respective standard counter part by a relatively large margin. Overvall, the best result achieved with using all available features was $F_1 \approx 0.614$ for the confSVM while the absolute best result was achieved by confSVM with leaving out the feature group lex-mpqa with $F_1 \approx 0.619$.

[20] See section 5.4.3.
[21] See section 5.2.1 for further details.
[22] See section 4.

Feature Group	confSVM	SVM	confMaxEnt	MaxEnt
uni	0.574	0.455	*0.578	0.369
bi	*0.517	0.382	0.514	0.202
tri	0.481	0.248	*0.483	0.243
quad	*0.408	0.189	0.403	0.176
sub2	0.491	0.432	*0.497	0.395
sub3	0.560	0.484	*0.564	0.423
sub4	**0.582**	**0.501**	***0.584**	0.426
w2v	0.538	0.467	*0.545	0.437
cmu	0.550	0.462	*0.555	0.377
lex-nrc	0.327	0.365	*0.448	0.377
lex-mpqa	0.368	0.373	*0.461	0.373
lex-tweetos	*0.476	0.443	0.473	0.430
lex-opinion	0.517	0.472	*0.519	**0.469**

Table 5.2: Results for using only a single feature group for classification. The results shown are the macro F_1 scores for three-class classification and confidence based rejection classification using the *related domain* for training and *in-domain test* for evaluation. The best score in each column is written in bold face, the best score in each row is indicated with an asterisk.

Leaving out the feature group lex-tweetos had the strongest impact on the best performers confSVM and confMaxEnt while SVM was affected most by leaving out w2v features and MaxEnt by removing lex-opinion. Performance of the lexicon features seems to vary by relatively large amounts depending on the source they were generated from. The manually created lexicon feature groups lex-mpqa and lex-opinion are mostly reducing the F_1 score of the well performing classifiers. Probably, this is the case because they were created with formally correct texts in mind and do not perform well for the informal language that often occurs in short texts such as tweets. Suprisingly, the lex-nrc lexicon features which also were generated from a tweet corpus have mixed effects on the performance. In two cases performance is increased when leaving them out, in one case it does only change marginally. Only for MaxEnt the use of the feature group actually increases the F_1 score. The lex-tweetos features have positive effects for all classifiers. Hence, it seems beneficial to regenerate a new lexicon for the application at hand if sufficient data from within the relevant time frame is available.

While most of the features have a relatively small impact compared to the overall performance, the total impact of using all feature groups with positive effects increases performance drastically over using only single features groups, as is shown in table 5.2.

5.3.5 Direct Approach

For the direct approach, *in-domain test* was used as both training and test data by using ten-fold cross validation. The reported F_1 score in this experiment is the mean value of all ten runs, the standard deviation is also reported. Results are presented in table 5.4.

Feature Groups	confSVM	SVM	confMaxEnt	MaxEnt
all	0.614	0.536	0.612	0.504
all \ {uni}	−0.0031	−0.0008	−0.0021	−0.002
all \ {bi}	−0.0097	−0.0017	+0.0015	−0.0016
all \ {tri}	−0.0010	−0.0001	+0.001	−0.0001
all \ {quad}	+0.0030	−0.0005	+0.0012	−0.0001
all \ {sub2}	−0.0058	+0.0009	+0.004	−0.0024
all \ {sub3}	−0.0006	−0.0024	+0.001	−0.0042
all \ {sub4}	−0.0005	−0.0028	+0.0016	−0.0040
all \ {w2v}	−0.0036	**−0.0086**	−0.0068	−0.0059
all \ {cmu}	−0.0052	−0.0024	+0.0003	−0.0009
all \ {lex-nrc}	−0.0001	+0.0017	+0.0025	−0.0021
all \ {lex-mpqa}	+0.0050	+0.0013	+0.0006	−0.0039
all \ {lex-tweetos}	**−0.0176**	−0.0006	**−0.0073**	−0.0059
all \ {lex-opinion}	+0.0025	−0.0034	+0.0038	**−0.0068**

Table 5.3: Deviation of the macro F_1 scores when leaving out one feature group at a time for three-class classification and confidence based rejection classification using the *related domain* for training and *in-domain test* for evaluation. The biggest decrease when leaving a feature out in each column is written in bold face.

The confidence based rejection variants of the classifiers do not dominate the standard variants as strongly as it was the case for the transfer learning approach. The overall best results were achieved by cmu cluster features in conjunction with SVM ($F_1 \approx 0.613 \pm 0.035$) and MaxEnt ($F_1 \approx 0.614 \pm 0.036$). However, many of the results do not differ significantly from one another as can be observed by examining the standard deviations.

Lexicon features seem to work well for the confidence variants of the classifiers but suffer severe performance drops for the standard variants. In conjunction with SVM the scores of lex-nrc, lex-mpqa and lex-tweetos are at around $F_1 \approx 0.01$ which is not feasible for classification. The best performing lexicon for SVM was lex-opinion - a handcrafted lexicon - with $F_1 \approx 0.280$ which is still lacking. Results for MaxEnt are slightly better, just lex-nrc and lex-mpqa have $F_1 < 0.2$ with lex-tweetos and lex-opinion having a score in the range of $F_1 \approx 0.35$.

The n-gram and subgram feature groups are mostly performing on similar orders of magnitude with unigrams taking the lead and quadgrams performing worse of all n-grams in all cases.

Many of the investigated feature groups yield quite feasible results $F_1 > 0.5$ for at least one classifier and can be recommended for usage.

The results of the leave-one-out experiment are presented in table 5.5. When using all available features the lead is taken by the standard versions of the classifiers with $F_1 = 0.659 \pm 0.038$ for SVM and $F_1 = 0.659 \pm 0.035$.

Word embedding features (w2v) had the strongest impact on all classifiers (decrease

Feature Group	confSVM	SVM	confMaxEnt	MaxEnt
uni	0.560 ± 0.027	*0.600 ± 0.039	0.557 ± 0.027	0.596 ± 0.038
bi	0.517 ± 0.024	0.461 ± 0.042	*0.518 ± 0.029	0.508 ± 0.035
tri	*0.501 ± 0.026	0.418 ± 0.045	0.499 ± 0.025	0.412 ± 0.043
quad	0.446 ± 0.041	0.331 ± 0.053	*0.447 ± 0.042	0.326 ± 0.053
sub2	0.494 ± 0.023	0.522 ± 0.028	0.498 ± 0.019	*0.527 ± 0.029
sub3	0.547 ± 0.024	0.576 ± 0.041	0.549 ± 0.026	*0.578 ± 0.032
sub4	0.567 ± 0.022	*0.600 ± 0.035	0.565 ± 0.023	0.597 ± 0.039
w2v	0.568 ± 0.023	0.577 ± 0.049	0.567 ± 0.022	*0.603 ± 0.033
cmu	**0.578** ± 0.021	**0.613** ± 0.035	**0.576** ± 0.021	**0.614** ± 0.036
lex-nrc	0.452 ± 0.014	0.114 ± 0.143	*0.453 ± 0.013	0.151 ± 0.151
lex-mpqa	*0.456 ± 0.014	0.087 ± 0.133	0.454 ± 0.012	0.116 ± 0.143
lex-tweetos	*0.507 ± 0.017	0.076 ± 0.153	0.506 ± 0.016	0.340 ± 0.009
lex-opinion	0.505 ± 0.013	0.280 ± 0.141	*0.506 ± 0.014	0.362 ± 0.011

Table 5.4: Results for using only a single feature group for classification. The results shown are the mean macro F_1 scores for three-class classification and confidence based rejection classification using the *in-domain test* dataset for training and evaluation with ten-fold cross validation. The best score in each column is written in bold face, the best score in each row is indicated with an asterisk.

$d \in [-0.0111, -0.0166]$) with the difference to all other feature groups being one or more orders of magnitude higher. Differences in F_1 scores of the other feature groups compared to using all features remain in similar value ranges.

Another notable observation is the effect of leaving out subgrams. In 9 of 12 cases leaving them out yielded an improvement in performance instead of a decrease.

Features based on the lexicons tend to improve the classification results with lex-tweetos seeming to have the greatest positive impact of all lexicons. The feature group lex-nrc performed worse of all lexicons; it decreased performance for 3 out of the 4 classifiers.

While the cmu cluster features performed well in isolation, their impact on the top configurations is rather limited. For SVM the F_1 score decreased by just -0.0001 when leaving them out and for MaxEnt it was -0.0022.

5.3.6 Summary and Conclusion

State-of-the-art manual feature engineering methods have been thoroughly investigated with regard to their feasibility for general purpose Twitter sentiment analysis. Two classifier variants have been investigated, namely direct three-class classification and confidence based rejection classification that is only trained with *positive* and *negative* training data. The classifiers were examined with two different datasets, the original *in-domain test* dataset for training and evaluation (direct approach) and a *related domain* dataset for training where evaluation was also done with *in-domain test* (transfer approach).

Feature Groups	confSVM	SVM	confMaxEnt	MaxEnt
all	0.608 ± 0.024	0.659 ± 0.038	0.604 ± 0.022	0.659 ± 0.035
all \ {uni}	−0.0027	−0.0022	−0.0008	+0.0025
all \ {bi}	−0.0025	−0.0062	−0.0003	−0.0025
all \ {tri}	−0.0013	−0.0030	−0.0003	+0.0007
all \ {quad}	+0.0001	−0.0025	−0.0013	+0.0007
all \ {sub2}	+0.0011	+0.0018	+0.0040	+0.0043
all \ {sub3}	+0.0011	+0.0073	+0.0018	+0.0060
all \ {sub4}	+0.0011	−0.0002	−0.0004	−0.0006
all \ {cmu}	−0.0014	−0.0001	+0.0007	−0.0022
all \ {w2v}	**−0.0111**	**−0.0133**	**−0.0166**	**−0.0140**
all \ {lex-nrc}	+0.0013	−0.0001	+0.0004	+0.0033
all \ {lex-mpqa}	+0.0018	−0.0022	−0.0004	−0.0010
all \ {lex-tweetos}	−0.0053	−0.0032	−0.0038	−0.0026
all \ {lex-opinion}	−0.0014	−0.0025	−0.0002	−0.0009

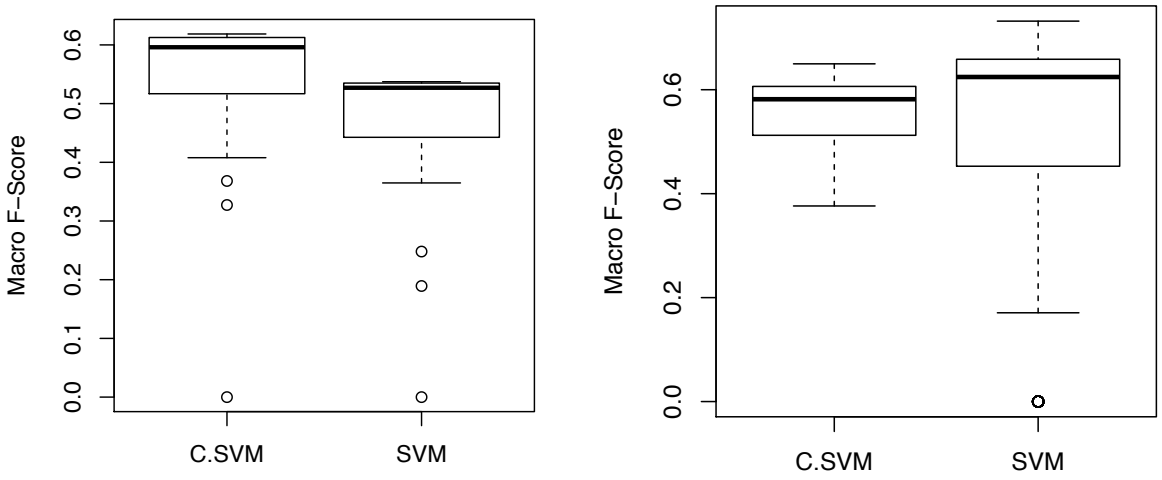
Table 5.5: Deviation of the mean macro F_1 scores when leaving out one feature group at a time for three-class classification and confidence based rejection classification using the *in-domain* for training and for evaluation with ten-fold crossvalidation. The biggest decrease when leaving out a feature in each column is written in bold face.

The results presented in the previous sections indicate that most of the investigated state-of-the-art features are feasible for classification. However, the performance of the respective feature groups varied between the direct approach and the transfer approach. Consequently, when attempting to transfer data from related Twitter sentiment analysis domains it can be recommend to reevaluate the feature groups since their efficiency may differ from domain to domain.

Furthermore, it can be stated that the confidence based rejection classifier variants perform consistently better for the transfer approach. While the classes *positive* and *negative* from the related domain dataset have a large overlap with the in-domain dataset, the class *neutral* differs more from *uncertain*. Hence, it is a more promising approach to train only with *positive* and *negative* and label tweets as *uncertain* when neither of the other two class can be reliably assigned. Another benefit of the confidence based classifiers is illustrated in figure 5.5b. While the variance in performance for the transfer approach is generally about the same for both classifier variants, the variance of the direct approach is much smaller for the confidence variants. Hence, confidence based classifiers seem to be less susceptible to the choice of features groups.

Moreover, it has to be noted that the performance of a feature group in isolation not necessarily indicates their performance when combined with other feature groups as was the case for the cmu cluster feature in the direct approach for example.

When using the direct approach with in-domain training data the confidence based rejection classifiers were still competitive but often outperformed by the direct approach which was able to achieve the overall best result of $F_1 \approx 0.659 \pm 0.038$. Having smaller amounts of in-domain data available seems to be more efficient than to transfer larger



(a) transfer approach **(b)** direct approach

Figure 5.5: Boxplots of all F_1 scores previously presented in this section for both variants of the SVM classifier for both experiments. The same plot for MaxEnt is qualitatively rather similar and is hence omitted here.

amounts from other domains. However, when unsure about which feature groups to use, one may still be on the safer side when using the confidence variants due to their lesser susceptibility to the choice of feature groups.

5.4 Deep Convolutional Neural Networks

Deep convolutional neural networks are mostly independent of manual feature engineering and are capable of learning useful features and internal representations in the training process[23]. Since creating and evaluating manually engineered features is a resource and time intensive process and the viability of the features may vary in time - as was the case for the automatically generated lexicons -, it can be beneficial to be able to rely upon approaches without the necessity of any manual intervention.

5.4.1 Direct Approach with Pretraining and Uniform Filter Size

The experiment described in this section[24] had the goal to evaluate the general viability of an archetypical deep convolutional neural network architecture as described in 2.2.4 but in a first step only using one filter size. Parameters that have to be set beforehand and cannot easily be learned as part of the training process are the dropout rate d, the activation function f and the configuration of the convolution filters. Since not all possible combinations of this parameters could be evaluated due to time constraints it was chosen to mainly investigate the minimum, maximum and median of the intervals suggested by Zhang and Wallace (2015). All combinations of the following parameter settings have been investigated:

[23] See section 2.2.4 for further details.
[24] Results will partly by published as Haldenwang et al. (2017).

- dropout $d \in \{0, 0.25, 0.5\}$
- activation function $f \in \{\text{linear}, \tanh, \text{ReLU}\}$
- filter sizes $s \in \{2, 3, 4, 5, 6, 8, 10\}$
- number of filters per filter size $n \in \{50, 325, 600\}$

Initialization of the weights was done by uniformly sampling values from the interval $[-0.05, 0.05]$. The word embedding layer was initialized with word embeddings of dimension 100 which were generated as described in section 2.1.2. This preinitialization with knowledge that can be gained in an unsupervised manner reliefs the training process of the burden to generate general semantic information; it just has to modify existing knowledge to fit the task of general purpose Twitter sentiment analysis. To a certain degree this approach reduces the necessity to have huge amounts of labeled data at hand.

Only minimal preprocessing is applied to reduce obvious noise and ensure consistency with the word embeddings. Preprocessors used were `AtMentionReplacer`, `URLReplacer` and `NonWordRemover`.[25]

Since deep convolutional neural networks often require large amounts of data to work well it was chosen to pretrain the network with related domain data. First, the network was trained with 1,000,000 distantly supervised tweets of *positive* and *negative* sentiment which were sampled from the dataset of Neubauer (2014). Moreover, the vocabulary of the network was restricted to the 200,000 most common words in that corpus. Additionally, the network was pretrained with all three classes from the manually annotated *related domain* dataset introduced in section 5.3, where *neutral* was treated as *uncertain*.

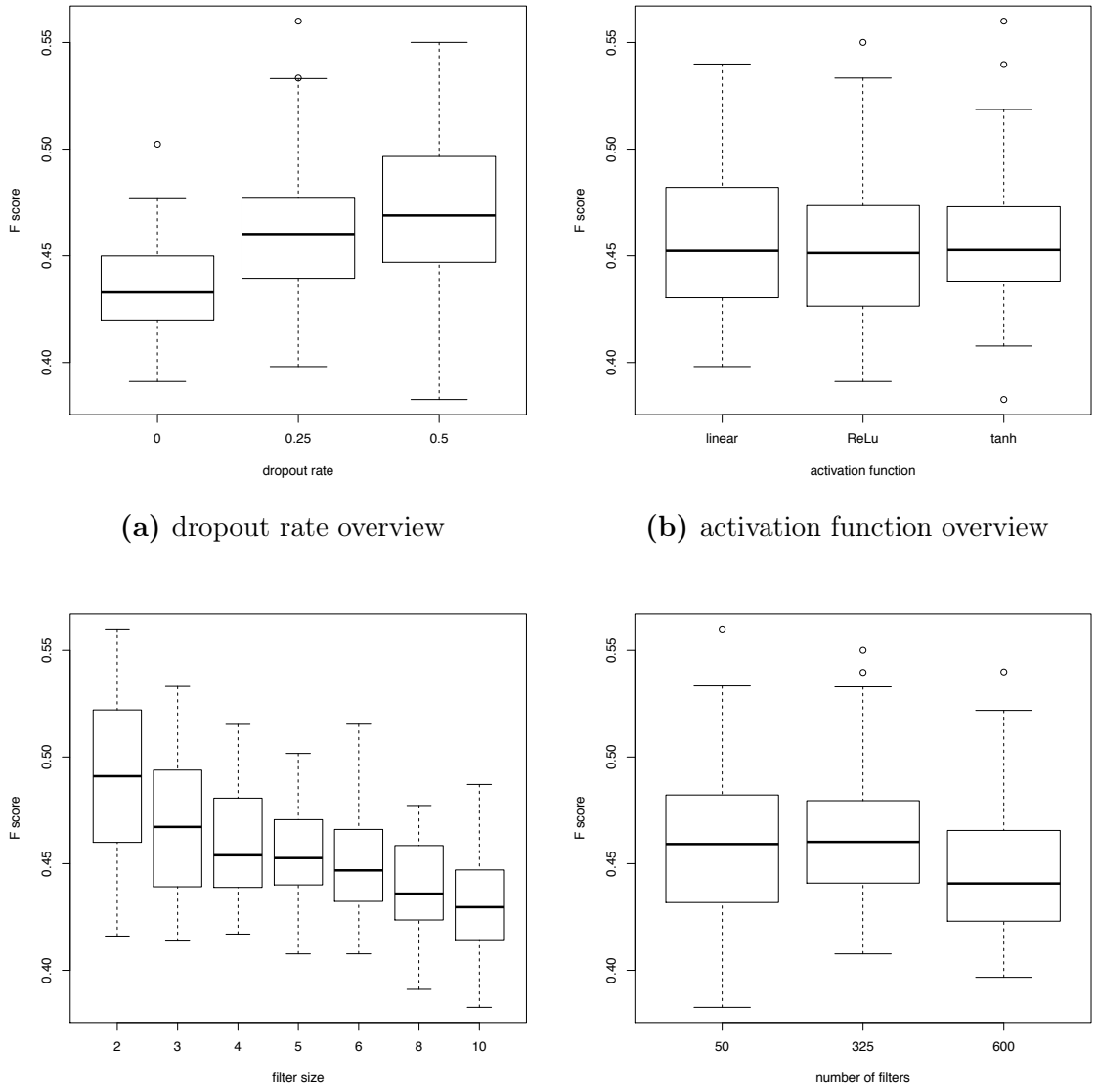
The training procedure is based on the suggestions of Zhang and Wallace (2015). Each training tweet is presented once to the network in a random order. Weight updates are taking place after the classifier processed a batch of 50 tweets. Updates are done with AdaGrad (Duchi et al. 2011) using cross-entropy between classes as cost function.

Finally, the network was evaluated with 10-fold cross-validation for each of the 189 parameter combinations on the dataset *in-domain test* that was introduced in section 5.3. The average F_1 score of the ten runs is reported as a result.

Overall, the best configuration was $d = 0.25$, $f = \tanh$, $s = 2$ and $n = 50$ with a score of $F_1 = 0.560 \pm 0.026$. To be able to assess the influence of the parameters that have been varied, a box plot of the results for each of the parameters is presented in figure 5.6 to summarize the results.

Looking at figure 5.6a it comes to attention that using a dropout rate of zero - i.e. using no regularization at all - can not be recommended. The performance is consistently sub par when compared to the dropout rates $d \geq 0.25$. It seems even the simplest deep convolutional neural network architecture is overfitting without regularization. Comparing the results for the other dropout rates to using no dropout it can be stated that dropout regularization seems to work well for the given architecture. While $d = 0.5$

[25] See section 2.1.1 for further details on preprocessing.



(a) dropout rate overview

(b) activation function overview

(c) filter size overview

(d) number of filters overview

Figure 5.6: Boxplots of all 189 parameter combinations which are showing the resulting F_1 scores by parameter value.

yields a generally better performance for most parameter configurations, the overall best result was achieved with $d = 0.25$.

In Figure 5.6b the overview by activation function is presented. The median of all three functions is very close to one another and they vary mainly in distribution of the scores. Another notable observation is the increased density of the results for the *tanh* activation function. It does not seem to be as susceptible to variance in the other parameters as its two counter parts. However, the overall best result and the overall worst result used *tanh* as activation function. Hence, the spread can be considered to be smaller than for the other activation functions but the farthest outliers can also be found there. In conclusion, all three activation functions yield similar performance and

should all be investigated when evaluating other network architectures.

A more obvious relation can be observed between performance and size of filters, as illustrated in figure 5.6c. The overall performance drops drastically with increasing filter size. This behaviour is another hint that the most important phrases for sentiment analysis are rather small and the words it consists of are often close together in the sentence. However, even at filter size 10 the absolute score is still feasible and should not be overlooked. For *filter size* the overall best result falls into the generally best performing filter size $s = 2$.

Unfortunately, the correct choice of the *number of filters* is not that obvious, as illustrated in figure 5.6d. While 325 filters show the least variance and better general performance than 600 filters, the overall best result is achieved with 50 filters. However, the worst overall results were also recorded when using 50 filters with a filter size of ten.

5.4.2 Combining Different Filter Sizes

As suggested by Zhang and Wallace (2015) it was investigated how combining different filter sizes affects the overall performance of the network. Since the process is very time consuming, it was chosen to use the best single-filter configuration (50 filters of size 2, $d = 0.25$, activation function *tanh*) that was discussed in the previous section, as a starting point. Additional filters of size $s \in \{3, 4, 5, 6, 8, 10\}$ are incrementally added to the configuration with the filter count $n \in \{50, 325, 600\}$. After evaluating a filter size s_i with all three counts n, the best result is used as a basis for s_{i+1} if it yielded an improvement over the previous configuration s_{i-1}. The network is trained and evaluated with ten-fold cross-validation on the *in-domain test* dataset. The mean F_1 score and the respective standard deviation for each evaluated configuration were reported.

Results are presented in table 5.6. Overall, adding additional filter sizes to the initial configuration yielded no improvement and resulted in consistent performance decreases. The drop in performance seems to increase with growing filter size s. Furthermore, it can be noted that the performance drop is bigger with higher filter counts. Probably this is the case due to overfitting problems with increasing number of weights to be learned. In summary, it can be said that adding additional filters to the formerly best network configuration can not be recommended for the direct 3-class approach. Despite this, it may be insightful to investigate more parameter sets in this manner but due to limited resources and the discouraging results of this experiment, the possibilities were not pursued any further in this work.

5.4.3 Confidence Based Rejection Classification

Conceptually, the approach is very similar to that described in sections 5.3.3 and 5.3.4. The classifier is trained with only *positive* and *negative* data from closely related domains and classifies samples as *uncertain* when it cannot reliably assign one of the other two classes. Since the neural network architecture used in this research outputs probability

Filter Configuration	F_1
2: 50,	**0.560 ± 0.026**
2: 50, 3: 50	*0.511 ± 0.059
2: 50, 3: 325	0.427 ± 0.090
2: 50, 3: 600	0.435 ± 0.090
2: 50, 4: 50	* 0.503 ± 0.063
2: 50, 4: 325	0.445 ± 0.087
2: 50, 4: 600	0.410 ± 0.098
2: 50, 5: 50	* 0.485 ± 0.073
2: 50, 5: 325	0.423 ± 0.091
2: 50, 5: 600	0.422 ± 0.086
2: 50, 6: 50	* 0.456 ± 0.073
2: 50, 6: 325	0.422 ± 0.090
2: 50, 6: 600	0.415 ± 0.108
2: 50, 8: 50	* 0.468 ± 0.078
2: 50, 8: 325	0.427 ± 0.093
2: 50, 8: 600	0.385 ± 0.096
2: 50, 10: 50	* 0.410 ± 0.104
2: 50, 10: 325	0.373 ± 0.113
2: 50, 10: 600	0.378 ± 0.111

Table 5.6: Mean cross-validation F_1 score with standard deviation for each investigated filter configuration. The notation for the filter configurations is to be understood as *filter size: number of filters*. The best result for each added filter size is annotated with an asterisk; the overall best result is written in boldface.

estimates p_x for the *positive* class and q_x for the *negative* class, the confidence for the classification result of a sample x can be computed as:

$$C(x) = 1 - \frac{\min\{p_x, q_x\}}{\max\{p_x, q_x\}} \tag{5.2}$$

During the training process a feasible uncertainty threshold t has to be determined so the classification can then be performed as:

$$class(x) = \begin{cases} \text{uncertain} & \text{for } C(x) < t \\ \text{positive} & \text{for } t > C(x) \wedge p_x > q_x \\ \text{negative} & \text{for } t > C(x) \wedge q_x > p_x \end{cases} \tag{5.3}$$

Based upon the results discussed in section 5.4.1, not all combinations of parameters are investigated, but only the most promising setup $d = 0.25$, $f = \tanh$, $s = 2$ and $n = 50$ with a previous score of $F_1 = 0.560 \pm 0.026$ for the direct approach. Architecturally, the network investigated in this experiment was almost identical, the only difference being the presence of just two output neurons instead of three. The network was initialized in the same manner as before but the training and evaluation process

Filter Configuration	t	F_1
2: 50	0.8	*0.584
2: 50, 3: 50	0.8	0.568
2: 50, 3: 325	0.9	0.592
2: 50, 3: 600	0.9	*0.595
2: 50, 3: 600, 4: 50	0.8	0.585
2: 50, 3: 600, 4: 325	0.9	***0.605**
2: 50, 3: 600, 4: 600	0.9	0.594
2: 50, 3: 600, 4: 325, 5: 50	0.9	*0.601
2: 50, 3: 600, 4: 325, 5: 325	0.9	0.589
2: 50, 3: 600, 4: 325, 5: 600	0.9	0.589
2: 50, 3: 600, 4: 325, 6: 50	0.9	0.577
2: 50, 3: 600, 4: 325, 6: 325	0.9	*0.592
2: 50, 3: 600, 4: 325, 6: 600	0.9	0.588

Table 5.7: F_1 score and confidence threshold t for the confidence based rejection approach for each investigated filter configuration. The notation for the filter configurations is to be understood as *filter size: number of filters*. The best result for each added filter size is annotated with an asterisk; the overall best result is written in boldface.

differs. Pretraining was performed as previously described with the same set of distantly supervised tweets. From the *related domain* dataset[26] the *neutral* class was left out here. Next, for each confidence threshold $t \in \{0.1, 0.2, \ldots, 0.8, 0.9, 1.0\}$ the network is trained with the aforementioned training data (in the same manner as presented in section 5.4.1) and then evaluated on the dataset *in-domain dev*[27] using the macro F_1 score as performance measure. The threshold t with the best result is then chosen to be used for classification. The final system - where classification is performed according to equation 5.3 - is evaluated on the dataset *in-domain test* and the macro F_1 score is reported.

After the uniform filter version has been evaluated, additional filter sizes were added successively as described in section 5.4.2 and were trained and evaluated as described above.

Results are presented in table 5.7. The confidence based rejection variant of the starting configuration seems to slightly outperform its counterpart from the direct approach with a score of $F_1 \approx 0.584$. Adding 600 filters of size 3 increased the performance to $F_1 \approx 0.595$. Another increase to $F_1 \approx 0.605$ could be achieved by adding 600 filters of size 4. Additions of filter sizes 5 and 6 yielded no further improvements. Hence, the filter configuration *2: 50, 3: 600, 4: 325* achieved the best result with $F_1 \approx 0.605$.

Another benefit of a classifier which uses estimated probabilities as output lies in the fact that the confidence threshold is directly interpretable. Notably, all confidence thresholds are relatively high with $t \geq 0.8$. This result matches the initial idea of reliable general purpose Twitter sentiment analysis which is also in line with the qualitative impression

[26] See section 5.3 for details.
[27] See section 5.3 for further details.

one gains from annotating a lot of tweets: The sentiment of a tweet is either very obvious or rather uncertain.

5.4.4 Summary, Conclusion and Outlook

Applying deep convolutional neural networks to Twitter sentiment analysis gained increasing attention in recent years. A basic architecture which is representative for the archetype has been thoroughly investigated. The best result for the direct approach scored $F_1 \approx 0.560 \pm 0.026$ and is outperformed by the confidence based rejection approach which was able to achieve a score of $F_1 \approx 0.605$. For future work it may be interesting to investigate more complex architectures with multiple convolution layers[28]. Another interesting investigation would be to explore additional numbers of hidden layers. However, due to the preliminary results being consistently and significantly sub par to those of the manual feature engineering approach discussed in section 5.3, it was chosen not to pursue those investigations in this work. Overall, the performance is generally feasible for a 3-class problem and can be applied in real-world scenarios.

5.5 Reducing the Annotation Effort for Deep Convolutional Neural Networks with Uncertainty Sampling Based Active Learning

Acquiring a sufficient amount of manually annotated data for the training of deep neural networks is very labor intensive since tuning hundreds of thousands of weights properly often requires large amounts of data. One possibility to deal with low amounts of manually annotated data is the use of *distant supervision* approaches based upon emoticons as originally introduced in Pak and Paroubek (2010). Distant supervision has already successfully been used in the training process of various deep learning architectures for Twitter sentiment analysis (Severyn and Moschitti 2015, Deriu et al. 2016, Xu et al. 2016).

While noisy labels based on emoticons provide a good starting point for the training of a deep learning system, it is probably beneficial to use manually annotated training data for the specific task to achieve satisfying results.

A common approach to reduce the manual effort is *active learning*. Settles (2010) summarizes the idea of active learning as follows: "[...] a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns [...]". Given a large corpus of unlabeled data points, the learner may choose the samples from which it hopes to gain the most insights. The labels of the chosen data points are queried from an *oracle*, in this case a human annotator. The remainder of this section describes a study that was conducted to assess the feasibility of various metrics for measuring the potential information gain for unlabeled samples and then choosing the samples that are to be annotated.[29]

[28] Deriu et al. (2016) successfully used such network configurations as part of an ensemble of networks.
[29] A subset of the following results will be published as Haldenwang et al. (2017).

5.5.1 Investigated Active Learning Strategies

As a strategy to query the best suited tweets to label for the network, it was chosen to investigate *uncertainty sampling*, a strategy originally devised by Lewis and Gale (1994) which is both easy to implement and understand and thus commonly used. With this strategy, each tweet is assigned an uncertainty value which defines how uncertain the network is in finding the correct label for the tweet. The most uncertain tweets are then chosen to be labeled.

For a problem with three (or more) classes such as general purpose Twitter sentiment analysis, there are different metrics available to calculate uncertainty for an arbitrary sample x and chose the most uncertain tweet x^* from the pool of unlabeled samples. These metrics differ in how many of the class probabilities they take into account. In the following a short description for each of the metrics is provided, which is based upon the remarks of Settles (2010). A more thorough introduction can be found in the literature survey of Settles (2010).

The *confidence* metric can be used to choose the tweet x^*_{LC} whose label the network is *least confident* about:

$$x^*_{LC} = \arg\min_x P_\theta(\hat{y}|x)$$

The confidence is defined as the probability that the class label $\hat{y}$ chosen by the network θ is correct as considered by the network itself (and as such is the highest of the three probabilities for the three class labels).

The *margin* metric also takes the second highest probability into account by calculating the difference between the probabilities of the two class labels $\hat{y_1}$ and $\hat{y_2}$ the network believes to be most likely correct:

$$x^*_M = \arg\min_x P_\theta(\hat{y_1}|x) - P_\theta(\hat{y_2}|x)$$

A tweet with a smaller margin would be considered to be of higher uncertainty since the network has difficulties choosing between the labels $\hat{y_1}$ and $\hat{y_2}$.

Finally, the *entropy* metric considers the probability for all class labels $\hat{y_i}$ to calculate the amount of informativity each tweet has to offer to the network:

$$x^*_H = \arg\max_x -\sum_i P_\theta(\hat{y_i}|x) \log P_\theta(\hat{y_i}|x)$$

Within the course of this experiment, the effect of these metrics were compared to find out which is most helpful for the use case at hand.

To speed up the labeling process, the tweets were queried and labeled in batches of size h. Otherwise, the network would have to be retrained after each tweet and the pool of unlabeled samples to chose from would have been reevaluated after each tweet for

choosing the next tweet to be annotated. Such long pauses between labeling samples were deemed infeasible because of the strain the approach would put on the human annotator. However, since the uncertainty values are not recalculated after picking a tweet for a batch, this could lead to the tweets in the batch being very similar to one another since they all may occupy the same uncertain region of the feature space. To avoid this problem, diversity was introduced as a second criterion to the querying process as described in Patra and Bruzzone (2012): First, the $m = 3h$ most uncertain tweets where chosen which were then reduced to h both uncertain and diverse tweets by clustering them with kernel k-means[30] into h clusters and picking the most uncertain tweet from each cluster. Clustering was done with the library *pyclust*[31] and the default parameters of the library have been used. Tweets were vectorized for clustering by computing the sum of all the word embeddings corresponding to the words the tweet consisted of.

Settles and Craven (2008) argue that methods based upon uncertainty sampling are often prone to selecting outliers for annotation. While an outlier may be the most uncertain sample in the pool of unlabeled samples it still often does not convey the highest amount of usable information. To remedy this drawback and increase the quality of the selected samples, they suggested to additionally consider the *information density* of a sample and compute the chosen sample x^*_{ID} as:

$$x^*_{ID} = \arg\min_x \phi(x) \cdot \left(\frac{1}{U} \sum_{i=1}^{U} sim(x, x_i) \right)^{\beta} \tag{5.4}$$

where U is the number of samples left in the pool of unlabeled samples, $\phi(x)$ is one of the previously described uncertainty measures and the function $sim(x, x_i)$ is the cosine similarity between the vectors x and x_i:

$$sim(x, x_i) = \frac{x \cdot x_i}{||x|| \cdot ||x_i||} \tag{5.5}$$

Vectorization of the tweets is done in the same manner as for the k-means clustering mentioned above. The parameter β controls the influence of the density weighting and was chosen to be 0.5 based upon the results from Settles (2008).

To paraphrase the approach in natural language one could say that the uncertainty of each unlabeled sample is weighted with its average similarity to all other samples which hopefully increases the chance to select samples for annotation that provide useful information about as many other samples as possible and minimizes the possibility of choosing outliers.

[30] See Zhang and Rudnicky (2002) for further details.
[31] `https://github.com/mirjalil/pyclust`, last checked 24.07.2017.

5.5.2 General Experimental Setup

All experiments used the best performing neural network discovered by the direct approach experiment described in section 5.4.1 as a basis for the evaluation but should be transferable to similar network architectures that mainly differ in the parameter choice. The network used 50 filters of filter size 2, the activation function *tanh* with a dropout rate of 0.25 and achieved a score of $F_1 \approx 0.560$ with 10-fold cross-validation on the *in-domain test* dataset.

For all experiments, the network was pretrained with a 1,000,000 sample from the distantly supervised dataset from Neubauer (2014). Since the main goal of this investigation is to explore the possibilities to reduce the amount of manual annotation necessary, no further manually annotated data is used for pretraining. The distantly supervised corpus can be acquired completely automatically.

The dataset that was used is *in-domain complete* which is the union of *in-domain test* and *in-domain dev*[32].

5.5.3 Simulated Active Learning for Parameter Estimation

Considering the fact that the manual annotation of tweets in the active learning process is a time consuming and tedious task it was chosen to restrict the parameter sets that were investigated. To get an estimate what set of parameters is worthy of investigation, active learning is simulated in a first step. The simulation is done by using the *in-domain complete* dataset for 10-fold cross-validation. Samples to train with are chosen according to the currently investigated active learning strategy from the pool of the labeled training folds which are not used for testing. The average F_1 score is reported as result. While this approach is not strictly representative for an actual active learning scenario due to the limited size of the pool to chose samples from, it nevertheless provides a much better approximation to a good set of parameters than guessing.

Choosing the Number of Training Iterations

Since the goal of the approach is to minimize the number of tweets that have to be manually annotated, it seems to make sense to increase the weight of each sample by running more than one training iteration per sample. To assess the feasibility of this approach, the simulated experiment was run on the initial network with the parameter values $i \in \{1, 10, 100\}$ for the iteration size. Moreover, the batch mode was not used here yet, the network was retrained after each sample is chosen and the remaining pool of tweets was reevaluated and was assigned new confidence scores. The random baseline was generated by randomly sampling from the pool, the rest of the process was identical.

The results of performing only one training iteration are depicted in figure 5.7. Evaluating the network that was pretrained with the distantly supervised data (none of the

[32]See section 5.3.1 for more details.

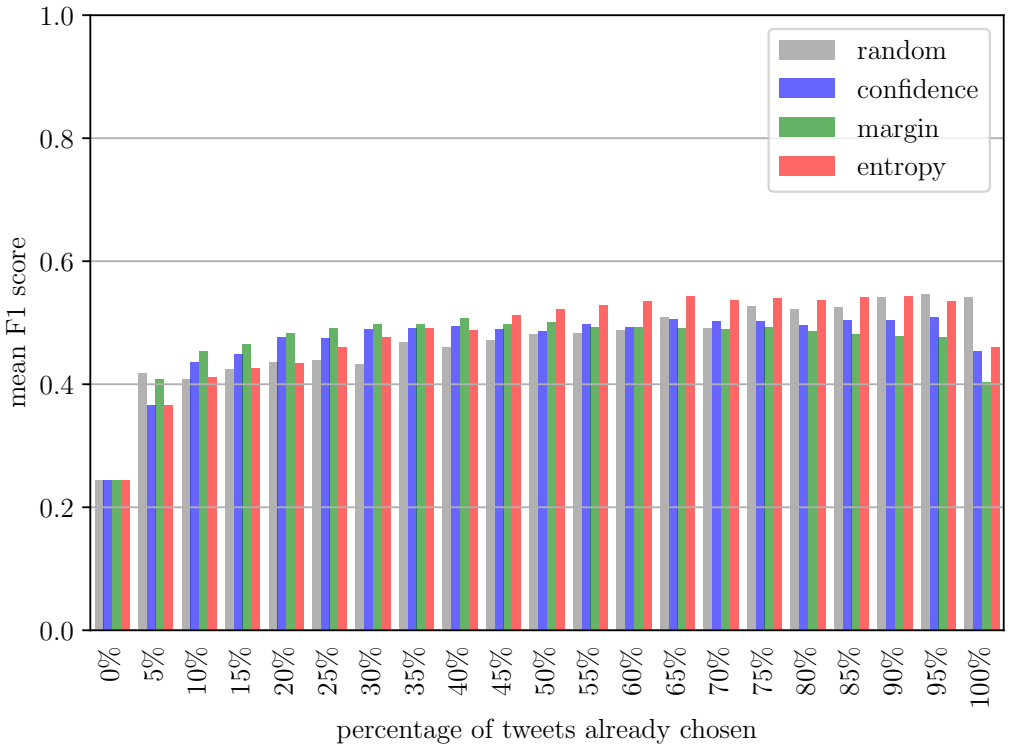


Figure 5.7: Visualization of the simulated active learning performance (F_1 score over the percentage of the pool that was used for training) for one training iteration per sample.

manually annotated data was used) revealed a very low performance that is below the score of just guessing the class. While this seems odd at first glance it makes sense when one considers that the distantly supervised tweets only consist of *positive* and *negative* examples and lacks examples for the class *uncertain*. Computing the results for just the polarity bearing classes yielded a score of $F_1 \approx 0.637$. However, not a single tweet of the class *uncertain* was classified as such which makes sense due to the fact the network has not ever seen a tweet of that class yet. While not yielding direct improvements for the three-class problem, the pretraining initialized the network with useful information about the polarity bearing classes and refined the underlying word embeddings accordingly. Adding only 5% percent of the samples from the pool yielded drastic performance improvements of about 100% when compared to using 0% of the manually annotated data. This drastic improvements further solidifies the viability of the pretraining step. Overall, the performance increases with increasing amounts of manually annotated data used for training. Up until about 50%-70% the active learning strategies tend to slightly outperform the random baseline. For more than 70% of manually annotated data used, the random baseline achieves similar scores and while the active learning metric's performance drops drastically at 100%, the random performance does not drop that strong. A possible explanation for the drop is that at the last evaluation step there are just the tweets left in the pool which the network is already rather certain about and training with them leads to overfitting of the parameters for such tweets which overwrites the generalization.

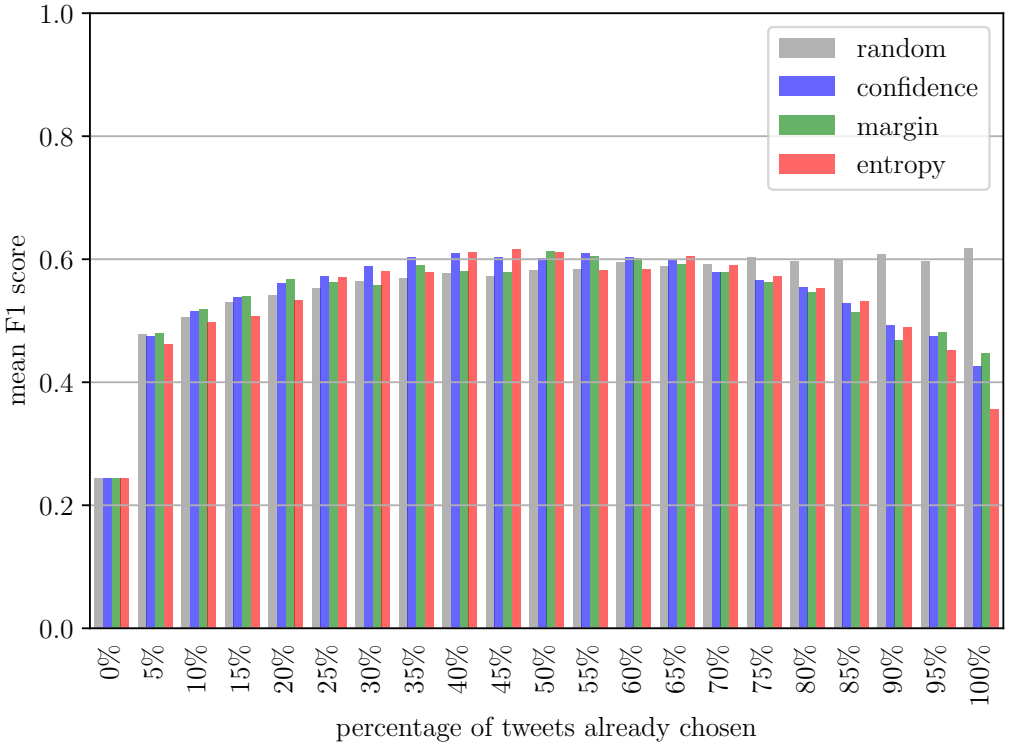


Figure 5.8: Visualization of the simulated active learning performance (F_1 score over the percentage of the pool that was used for training) for ten training iteration per sample.

Figure 5.8 depicts the results of using ten training iterations for each sample. First, it comes to attention that the general performance is significantly better with ten training iterations when compared to using just one iteration and performance also increases when adding data up to a certain percentage. Second, it can be stated that the negative effects that were observed in the one-iteration results seem to have been amplified by using ten training iterations. The performance drop of the active learning strategies is steeper than before and starts at about 70% of data used. Again, up until the point where performance starts to deteriorate, the active learning strategies slightly outperform the random baseline.

The results for the 100-iteration setup are presented in figure 5.9. Qualitatively, the results are very similar to those of using ten training iterations. The drop starts at about the same percentage of tweets used for training but its magnitude is worse than for ten iterations. Moreover, the general performance has slightly degenerated. No configuration managed to score above $F_1 = 0.600$ anymore.

Another interesting observation that can be made across the three result sets is the variance in performance of the random baseline. While the intermediate steps obviously are not comparable due to each training process using a different set of tweets, all 100% results should ideally be the same. However, that is not the case and only further solidifies that the order of using samples for training matters and choosing samples in the best possible order is an important step towards good performance.

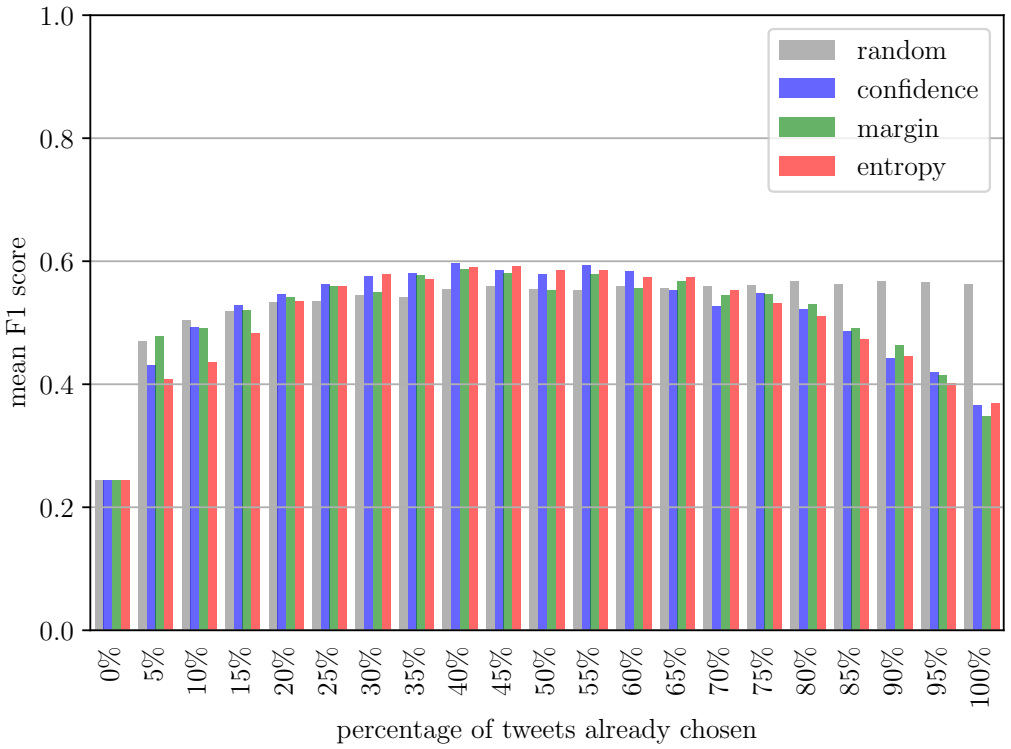


Figure 5.9: Visualization of the simulated active learning performance (F_1 score over the percentage of the pool that was used for training) for 100 training iteration per sample.

Using ten iterations over one seemed to increase performance and make the effects of the active learning strategies more visible. Increasing to 100 iterations only made results worse. Hence, it was chosen to continue the investigation with ten training iterations.

Preliminary Evaluation of the Batch Strategy

To assess the general viability of the batch strategy that was introduced in section 5.5.1, the approach was also simulated with the *in-domain complete* dataset using the same approach as in the previous section. The only difference was that $h = 20$ tweets were chosen with the batch mode strategy from $m = 60$ of the most uncertain tweets and the network was then trained with those 20 tweets before reevaluating the pool of unlabeled tweets. With informal experimentation it was determined that batches of 20 are a good compromise for the human annotators so that they are able to stay focussed and do not get bored because of long wait times.

Figure 5.10 consists of a visualization of the results. Qualitatively, the results seem very similar to not using the batch mode. The overall performance suffered a slight but non-significant drop when compared to the results presented in figure 5.8. However, the magnitude of the performance drop is not as steep. A possible explanation could be that at the end of the process more of the uncertain tweets are left due to the added diversity criterion. This seems to mitigate the overfitting to some degree. Based upon this results it was chosen to perform further experiments with the batch mode because the usability

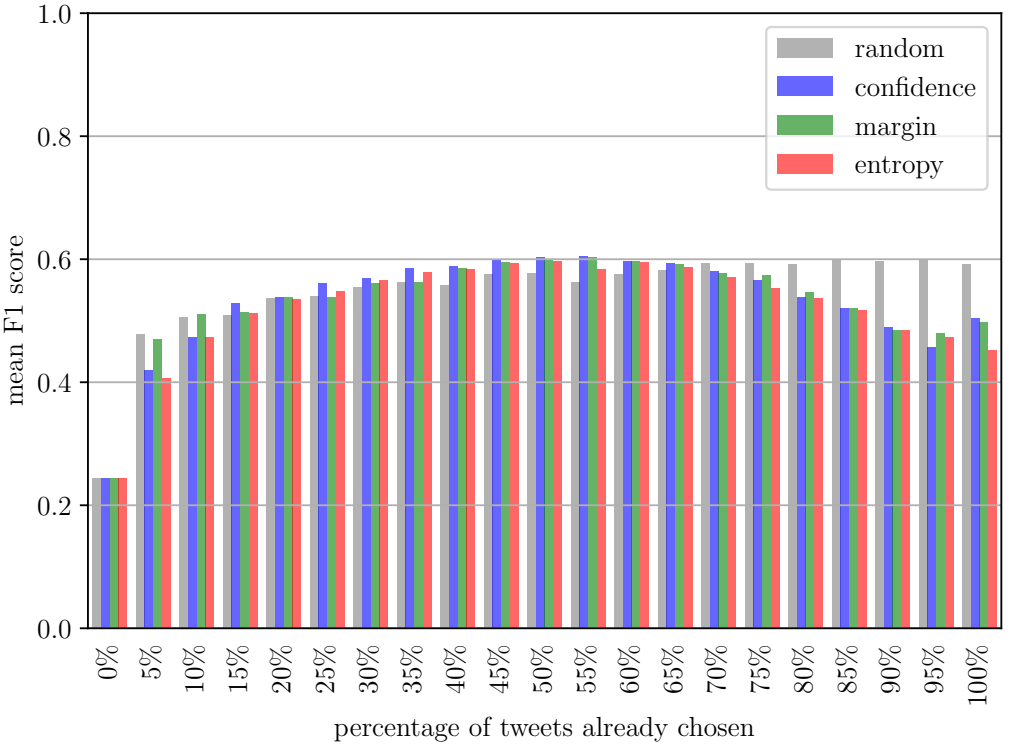


Figure 5.10: Visualization of the simulated active learning performance (F_1 score over the percentage of the pool that was used for training) for ten training iterations in batches of size 20.

for the human annotators is much more reasonable while not loosing significant amounts performance.

Preliminary Evaluation of Density Weighting

The results which are presented in figure 5.11 were generated with ten training iterations and batch mode was applied, as discussed previously. The new addition was the density weighting approach which was hoped to enforce the maximum information gain per batch. The results seem to be very similar to those presented in figure 5.10. However, notable is a further reduction in the performance drop and significant recovery of the performance in the last two steps with 95% and 100% of the data. Density weighting seems to indeed be able to mitigate some of the problems from the previous configurations but did not lead to an overall performance improvement. Since the results are so similar it was chosen not to discard the approach yet and also investigate it with actual manual annotation of unlabeled samples that were chosen from a large corpus.

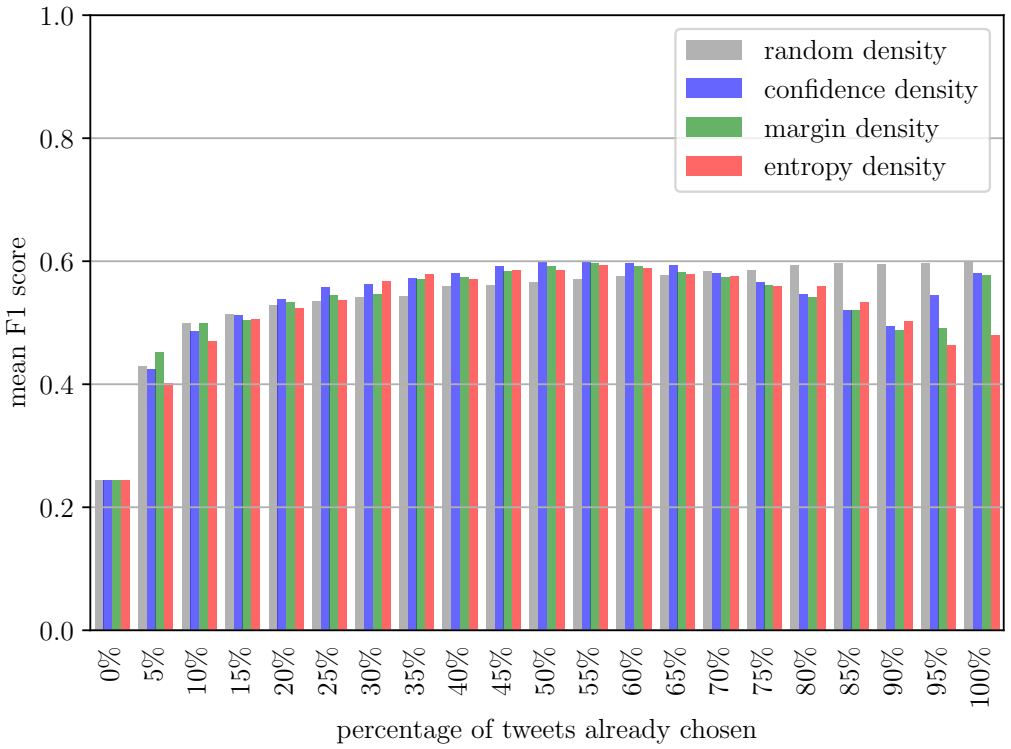
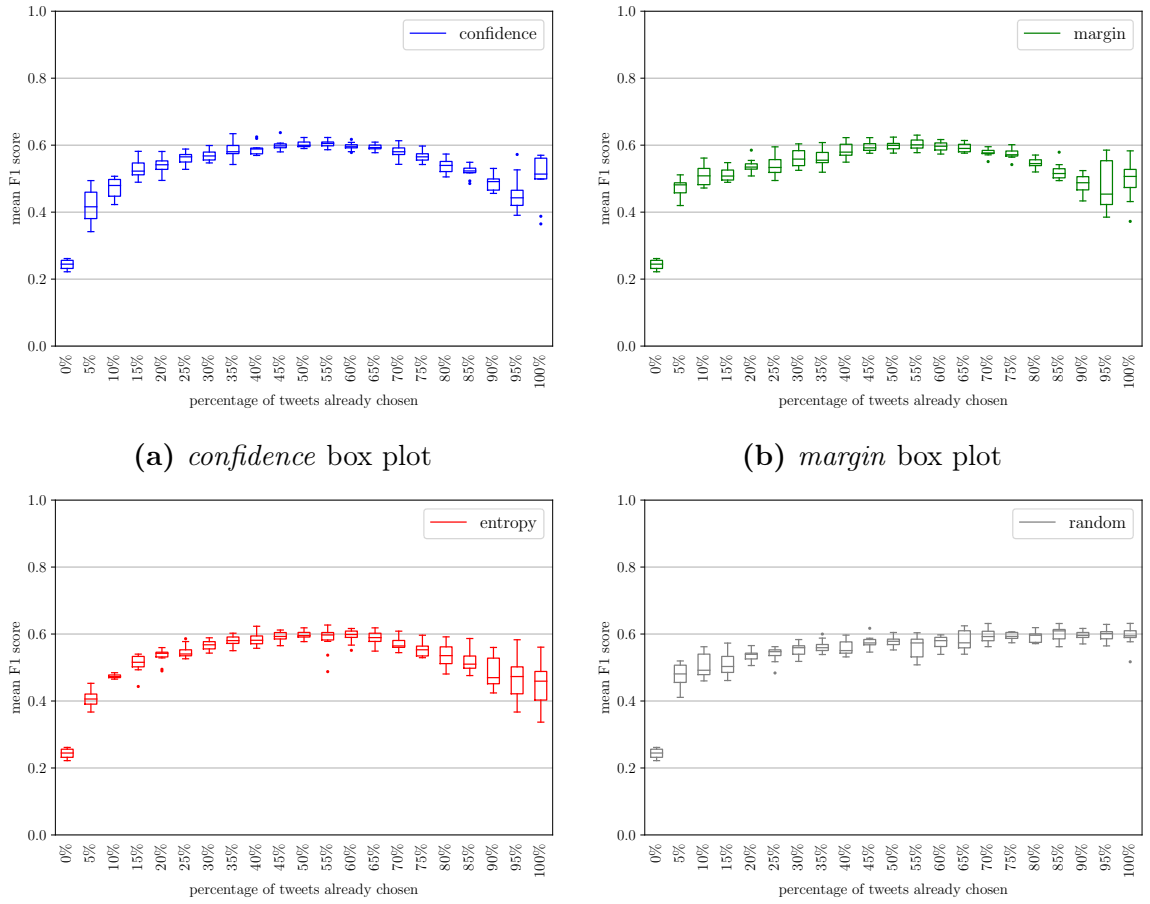


Figure 5.11: Visualization of the simulated active learning performance (F_1 score over the percentage of the pool that was used for training) for ten training iterations in batches of size 20 with density weighting.

A Closer Look at the Overfitting Behaviour

For further assessment of the aforementioned overfitting theory, the intermediate results of the cross validation runs were illustrated as a box plot which is presented in figure 5.12. The variance of the random baseline appears to be not that strong and fluctuating in a random manner. The behaviour of the uncertainty sampling metrics is qualitatively quite similar. Variance is higher when using smaller amount of data, converges to a very dense distribution at the peak performance and increases again with the performance drops. However, it should be stated that the *entropy* metric seems to be less susceptible to lack of data in the beginning while the *confidence* metric is least susceptible in the end of the process.

Conclusions to draw from these results are mainly that certain amounts of data are needed for proper generalization and that the aforementioned theory of overfitting problems when training with examples of high certainty at the end is a problem one has to deal with. However, this problem mainly occurs due to the properties of the simulation, mainly the rather small size of the sample pool to choose from. In a real active learning scenario this problems would probably be irrelevant.



(a) *confidence* box plot

(b) *margin* box plot

(c) *entropy* box plot

(d) *random* box plot

Figure 5.12: Box plots for the data originally presented in figure 5.10.

5.5.4 Active Learning Investigation with a Large Sample Corpus and Manual Annotation

Considering the results presented in the previous section, it was chosen to investigate two configurations of the strategy for all three uncertainty metrics. Both strategies use the initial network, that was pretrained with the distantly supervised data. The network is trained with 10 training iterations and the aforementioned batch mode with a batch size of $h = 20$ using the manually annotated data which is generated within the experiment. The first configuration does not use density weighting while the second configuration does apply it.

Interaction with the human annotator is done through a command line interface. After starting the experiment the network is initialized as described previously. Next, a corpus of 100,000 tweets[33] is classified by the initial system and 20 tweets are chosen according to the currently investigated configuration. The annotator then has to manually assign a label to each of the 20 tweets. After the annotation was completed, the network was

[33]The corpus was randomly sampled from the 33 million dataset of Neubauer (2014). Corpus size had to be reduced to maintain reasonable wait time for the human annotators.

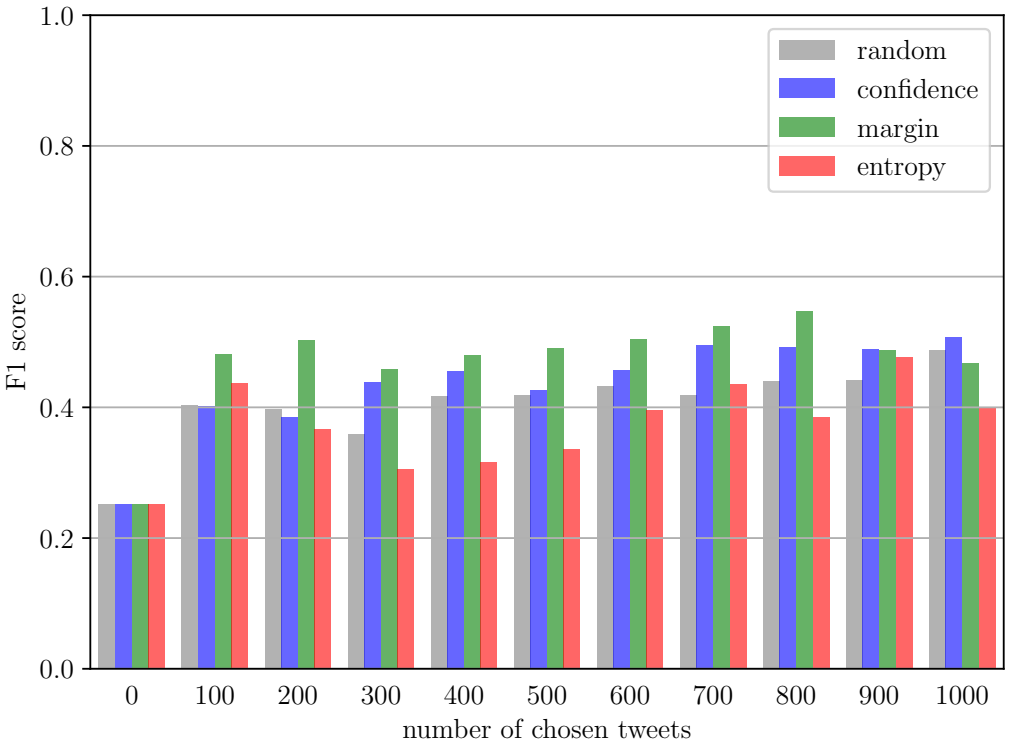


Figure 5.13: Results of using ten training iterations and batch mode

trained with the new batch and the remaining corpus was reclassified. This procedure was repeated until 1,000 tweets per setup were annotated.

The random baseline is generated by uniformly sampling from the dataset *in-domain dev* which consists of randomly selected, manually annotated tweets. *In-domain test* is used to evaluate the performance of the networks. There is no need for cross-validation since none of the data from *in-domain test* is used in the training process. The single macro F_1 score that was achieved on the test set is reported as result.

Figure 5.13 depicts the performance without applying density weighting. The random baseline yields solid results but seems to always be outperformed by either the *confidence* or *margin* metric. The *entropy* metric performs worse than random in almost all cases. Moreover, it seems to be the most unstable with the strongest fluctuations in performance. While the *margin* metric takes the lead for the first 800 annotated tweets, its effectiveness drastically drops at 900 and 1,000. Below 800, the *confidence* metric performed consistently worse than the *margin* metric but does not seem to suffer as severe a performance drop and at 1,000 labeled tweets takes the lead. Overall, the best performance achieved without density weighting was $F_1 \approx 0.547$ by the *margin* metric at 800 manually annotated tweets. Compared to the best result of $F_1 \approx 0.513$ from the data shown in figure 5.10 when using about the same amount of data (which is at 15%), it can be stated that the use of a large sample corpus with actual manual annotation yields notable improvements over the random baseline and the simulated active learning approach.

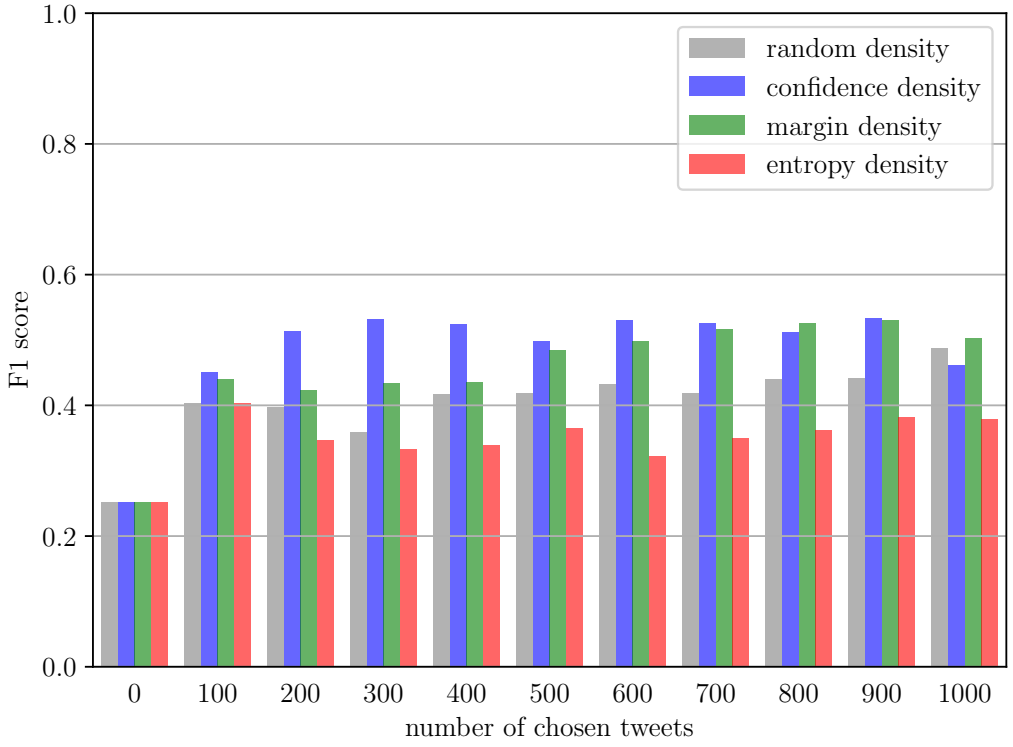


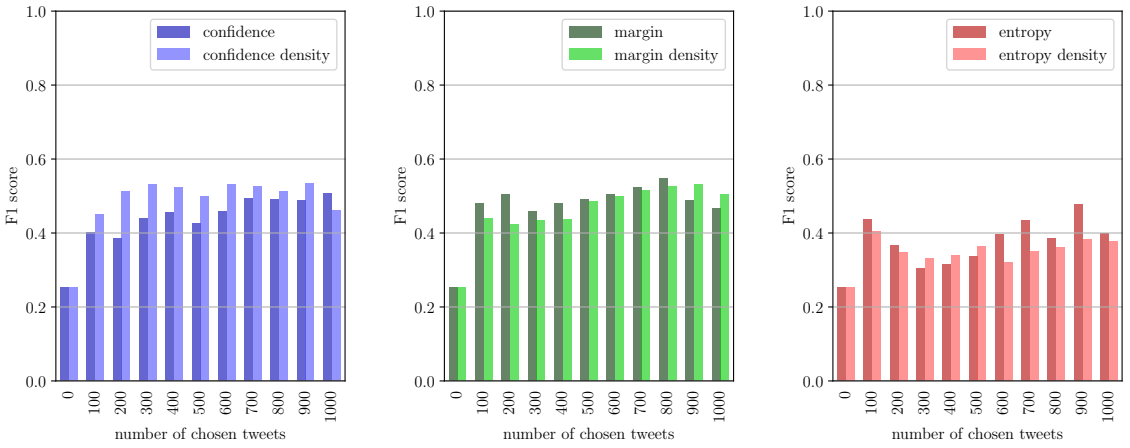
Figure 5.14: Results of using ten training iterations, batch mode and density weighting.

Adding density weighting to the setup yielded the results presented in figure 5.14. With density weighting, the dominating uncertainty metric was *confidence*. Especially at lower numbers of annotated tweets, the *confidence* metric significantly outperforms all other setups. With increasing numbers of annotated tweets the differences between the *margin* metric and the *confidence* metric became rather marginal. However, one of the two always outperformed the random baseline. Overall, the best result of the experiment with density weighting was achieved at 300 manually annotated tweets with a score of $F_1 \approx 0.532$.

A more direct comparison between the settings with and without density weighting is shown in figure 5.15. As indicated by the previous visualizations, the *confidence* metric performs better in conjunction with density weighting in almost all cases. The *margin* metric seems to work best without density weighting. *Entropy* performs better with density weighting in a few cases but the best performance was achieved without it. Moreover, both setups for *entropy* are consistently sub par when compared to the other two metrics. While the best overall performance was achieved by *margin* without density weighting, the best general performance when considering all intermediate evaluations could be observed for *confidence* with density weighting.

Error Analysis

Since the performance of the various setups was often rather close to one another it was chosen to take a closer look at the evaluation step with the overall best performing



(a) *confidence* **(b)** *margin* **(c)** *entropy*

Figure 5.15: Comparison of the setup with and without density weighting for each metric.

setup, using no density weighting and 800 manually annotated tweets, which was first presented in figure 5.13. Table 5.8 consists of the confusion matrix for each metric.

The network trained with the *margin* metric yielded the overall best performance with $F_1 \approx 0.547$. Looking at its confusion matrix it can be stated that the approach seems rather balanced. It has the highest average recall for the polarity bearing classes but is on the lower end for tweets of *uncertain* sentiment. Moreover, the confusion between *positive* and *negative* sentiment is rather high when compared to the other metrics, but still is balanced, the error is distributed almost evenly between the two classes. When analyzing large numbers of tweets - which is often the case due to the large volume of data in social networks such as Twitter - the confusion between positive and negative may even itself out statistically. However, the network classifies a large number of tweets as *positive* or *negative* that should have been classified as *uncertain*. Note that the amount of actually *uncertain* tweets that has been classified *negative* is considerably higher than the number of *uncertain* tweets which have been classified *positive*. Furthermore, the number of *negative* tweets classified as *uncertain* is almost double the number of *positive* ones classified *uncertain*. Hence, the system's confusion between *negative* and *uncertain* seems to be much higher than that between *positive* and *uncertain*.

Confidence was the second best performing metric with $F_1 \approx 0.491$. The balance that could be observed for *margin* could not be verified here. Mainly, the difference lies in the amount of correctly classified polarity bearing tweets, which is not only much lower than for *margin* but also imbalanced. The *positive* tweets are about three times as many as the *negative* ones. Moreover, the false positives for the polarity bearing classes are weighted the other way around, the network trained with *confidence* tends to classify two times as many *positive* tweets as *uncertain* than *negative* ones while the overall amount of sentiment bearing tweets classified as *uncertain* is smaller than for *margin*. However, the total amount of confusion between *positive* and *negative* is only about half of that of the *margin* metric.

The third rank was taken by the *random* baseline with $F_1 \approx 0.473$ whose error char-

	pos	neg	unc
pos	263	5	204
neg	172	752	1259
unc	745	867	3663

(a) *random.*

	pos	neg	unc
pos	643	74	583
neg	24	211	284
unc	513	1339	4259

(b) *confidence.*

	pos	neg	unc
pos	620	100	617
neg	93	685	844
unc	467	839	3665

(c) *margin.*

	pos	neg	unc
pos	172	16	92
neg	10	130	148
unc	998	1478	4886

(d) *entropy.*

Table 5.8: Confusion matrix for evaluation step at 800 annotated tweets from figure 5.13. Rows denote the label assigned by the neural network and columns represent the true label that was assigned by the human annotators.

acteristics, again, differ from the ones previously discussed. Results are even more unbalanced than those of the *confidence* metric, but the roles of *positive* and *negative* are reversed. Moreover, the *random* baseline shows a strong bias towards the *negative* class. Over 97% of the confusion between *positive* and *negative* - which is about the same order of magnitude as for *margin* - are *positive* tweets which actually have been classified as *negative* and 86% of actual *uncertain* tweets have been falsely classified as *negative*.

Entropy took the rear with a score of $F_1 \approx 0.385$. Nevertheless, it is notable that the absolute amount of the critical confusion between *positive* and *negative* is the smallest of all metrics. Moreover, the amounts of tweets that are actually *uncertain* but are classified as *positive* or *negative* is the smallest of all metrics. The amount of *positive* and *negative* tweets that have been classified *uncertain* is overall the highest. While the amount of correctly classified *positive* and *negative* tweets is the lowest of all metrics, *entropy* may be still be feasible for applications where it is important to keep the classes *positive* and *negative* as clean as possible and rather classify uncertain samples as *uncertain* when in doubt.

5.5.5 Harnessing the Generated Data to Improve the Initial Network

As a by-product of the aforementioned experiments, a corpus of sophisticatedly chosen, manually annotated samples consisting of 5,849 tweets was generated. Note that the experiments with manual annotation seemingly made use of 6,000 tweets, 1,000 for each uncertainty metric with and without density weighting. However, there was an overlap of the chosen tweets in the experiments and 5,849 is the count of unique tweets. The dataset consists of 1,613 *positive*, 1,713 *negative* and 2,523 *uncertain* tweets. One possible explanation for the higher share of *uncertain* tweets could be that the pretraining with *positive* and *negative* tweets resulted in less necessity for manual annotation for those classes. The dataset will be later referenced as *active learning complete*.

training data	macro F_1
distant supervision	0.252
distant supervision + related domain	0.548
distant supervision + active learning complete	0.590
distant supervision + related domain + active learning complete	0.627

Table 5.9: Performance of the initial network configuration when trained with various datasets, ordered by performance.

A comparison of the performance of the initial network when trained with the various available datasets is presented in table 5.9. The test set on which the networks were evaluated was *in-domain test*. As discussed previously, pretraining with the distantly supervised dataset yielded a poor score of $F_1 \approx 0.252$ since there were no samples for the *uncertain* class. Additionally training the network with the *related domain* dataset yielded improvements to a score of $F_1 \approx 0.548$ which could already be considered feasible for a three-class problem. However, training the system with the *active learning complete* dataset previously introduced in this section yielded a score of $F_1 \approx 0.590$. This further solidifies the point of this thesis that commonly used datasets such as *related domain* do not properly reflect the constraints of general purpose Twitter sentiment analysis. Moreover, it should be noted that the best performing system using the randomly annotated data directly only achieved a score of $F_1 \approx 0.560 \pm 0.026$ in ten-fold cross-validation and, hence, seems to be also outperformed. When combining *distant supervision* with *related domain* and *active learning complete* the performance of $F_1 \approx 0.627$ outperformed the best deep learning system - confidence based rejection - which managed to achieve a score of $F_1 \approx 0.605$.

In conclusion it can be stated that sophisticatedly chosen samples for manual annotation seem to be of higher quality and increased informativity for training a deep learning architecture than data transferred from a related domain or randomly selected samples.

5.5.6 Summary and Conclusions

After estimating a reasonable set of parameter setups with a simulated active learning approach, the uncertainty sampling metrics *margin*, *confidence* and *entropy* were investigated with a batch annotation approach and with and without density weighting applied. The overall best performing configuration was training the network with the *margin* metric using ten training iterations and 800 hand annotated samples and yielded a score of $F_1 \approx 0.547$. Compared to the score ($F_1 \approx 0.560$) of the initial network when pretrained with the *related domain* dataset that consists of about 25,000 manually annotated tweets and also using 8,000 tweets from *in-domain test* for training (with cross-validation), it is a notable result that the performance of the network trained with active learning achieves a comparable result with a only fraction of the amount of manually annotated training data. While those results are no direct improvement of the best system that was found in this work, they provide a useful starting point for investigating more complex deep learning architectures for future projects. It seems to

Feature Group	SVM	MaxEnt
uni	0.879 ± 0.023	0.892 ± 0.021
bi	0.877 ± 0.020	0.873 ± 0.023
tri	0.869 ± 0.025	0.866 ± 0.027
quad	0.856 ± 0.029	0.852 ± 0.033
sub2	0.862 ± 0.016	0.875 ± 0.018
sub3	0.865 ± 0.013	0.884 ± 0.021
sub4	0.868 ± 0.021	0.894 ± 0.018
w2v	0.879 ± 0.044	**0.900** ± 0.019
cmu	0.861 ± 0.017	0.873 ± 0.018

Table 5.10: Results of the single feature group experiment on spam classification. The average F_1 score of the cross-validation along with the standard deviation across runs is reported. The best result is written in bold face.

be an efficient approach to find a well performing network architecture using related domain data or small amounts of randomly selected data and then further refining the system with active learning rather then directly annotating huge amounts of random data which yields no additional insights for the system after a certain point.

Moreover, it could be shown that the data generated within the course of the active learning experiment can be used to train a network that outperforms networks trained with related domain data or randomly chosen in-domain data and are able to achieve a score of $F_1 \approx 0.627$ which is almost in line with the best result of the manual feature engineering approach.

5.6 Spam Classification

In a first attempt to distinguish the class *spam* from *non-spam* (= *positive* ∪ *negative* ∪ *uncertain*) it was investigated which of the previously used feature groups[34] that are not specific to sentiment are feasible for spam classification.

The classifiers investigated were SVM[35] and MaxEnt[36]. A combination of the datasets *in-domain test* and *in-domain dev* was used to train and evaluate the classifiers with ten-fold crossvalidation. Same as in the sentiment experiment[37] each feature group is first evaluated in isolation and then later all feature groups are combined and each is left out once to asses its contribution to the overall result.

Results of the single feature group experiment are presented in table 5.10. All of the investigated feature groups yielded feasible results with w2v taking the lead with $F_1 \approx 0.900 \pm 0.019$ when used in conjunction with the MaxEnt classifier.

[34] See section 5.3.
[35] See section 2.2.2.
[36] See section 2.2.3
[37] See section 5.3.

Feature Groups	SVM	MaxEnt
all	0.904 ± 0.019	0.907 ± 0.018
all \ {uni}	0.906 ± 0.018	0.907 ± 0.018
all \ {bi}	0.904 ± 0.021	0.905 ± 0.019
all \ {tri}	0.903 ± 0.020	0.905 ± 0.020
all \ {quad}	0.899 ± 0.020	0.904 ± 0.021
all \ {sub2}	0.908 ± 0.019	**0.911** ± 0.017
all \ {sub3}	0.904 ± 0.017	0.908 ± 0.020
all \ {sub4}	0.908 ± 0.019	0.908 ± 0.018
all \ {w2v}	0.881 ± 0.060	0.902 ± 0.020
all \ {cmu}	0.907 ± 0.019	0.908 ± 0.018

Table 5.11: Results of the leave-one-out experiment on spam classification. The average F_1 score of the cross-validation along with the standard deviation across runs is reported. The best result is written in bold face.

The results of leave-one-out experiment are presented in table 5.11. When combining all features the performance is slightly increased compared to the best single feature group result to $F_1 \approx 0.904 \pm 0.019$. The overall best result was achieved by the MaxEnt classifier when leaving out the feature group sub2 with $F_1 \approx 0.911 \pm 0.017$. Most feature groups had just marginal effects when left out, the notable exception being w2v which lead to the biggest performance drops.

While spam classification for more formal texts such as emails[38] is a well developed field, it was not necessarily obvious to be similarly easy to solve for the informal language used in social media. However, feasible results can be achieved by using relatively simple features directly based upon the words the tweets consist of. Both of the investigated classifiers performed in a very similar manner with MaxEnt having a slight edge over SVM.

Since the results of the previously described experiment were already feasible, more advanced techniques[39] for spam classification have not been further investigated and the best configuration is recommended for usage, due to the main focus of this work being the sentiment analysis.

5.7 Summary and Conclusions

After establishing the problem of reliable general purpose Twitter sentiment analysis and dissociating it from related Twitter sentiment analysis problems, a high quality dataset was introduced to be able to assess possible solutions to the problem with regard to their feasibility. Next, two archetypical approaches and a multitude of settings for them have been investigated, among those a rather novel approach of confidence based rejection

[38] See for example Yu and Xu (2008).
[39] See for example Mccord and Chuah (2011).

classification, which is able to solve the three-class problem while being trained with only two classes.

One of the archetypical approaches is using manual feature engineering in conjunction with various classification algorithms. It could be shown that transferring datasets from a related domain is consistently sub par to harnessing the new dataset which further solidified the claim that reliable general purpose Twitter sentiment analysis is not as similar to other Twitter sentiment analysis domains as it seems at the first glance. Furthermore, the confidence based rejection approach which also prominently tries to harness related domain data is consistently outperformed by the direct approach.

The second archetypical approach from the current state-of-the-art in related fields is the use of deep convolutional neural networks where no manual feature engineering is necessary. A commonly used basic architecture was closely investigated but yielded consistently sub par results compared to the manual feature engineering approach. Nevertheless, in case of the neural networks, the confidence based rejection approach yielded slightly better performance than the direct three-class approach. Moreover, it was investigated how the labeling effort for training deep convolutional neural networks in this setting can be reduced with active learning. The active learning strategies consistently outperformed the approach of randomly labeling samples and was able to achieve similar performance to the previously best system but with only a fraction of the amount of data used. Finally, all the additionally labeled data that was generated was used to train a network which topped all the previous network's performances significantly and which came close to the performance of the manually feature engineered system.

Reliably filtering *spam* was discovered to be easily done with a canonical approach.

All in all, it was shown that the most popular dataset from a closely related domain does indeed not match the novel problem of reliable general purpose Twitter sentiment analysis. While the data was useful, better results were achieved in most cases by using in-domain data of the new problem. However, the general approaches which were transferred from the related state-of-the-art seem to generally work well in the new setting with manual feature engineering yielding the best performance.

6 A Proof-of-Concept Architecture of a Real-Time Event Detection System with Explorative Data Analysis Capabilities

Due to the fact that this thesis was written in the practical oriented Media Computer Science Group it was chosen to provide at least a proof-of-concept architecture of a common application for general purpose Twitter sentiment analysis that was actually implemented and shown to be working practice - even though the implementation effort is not the main focus of this work.

In the following, the outline of the main modules of a distributed real-time event detection system with explorative data analysis capabilities is presented and discussed. The goal of the project was to provide a basic blueprint on how such a system can be designed, highlight the main challenges and problems one has to deal with and deliver a working prototype. Because of the project not being the main focus of this dissertation, not all aspects of the system could be systematically and thoroughly investigated. It was partly necessary to rely on informal experimentation and qualitative analyses to complete the project with the available resources, since the topic of real-time event detection on the public Twitter stream could fill one or more dissertations on its own.

Other constraints of a rather practical nature were the need to be able to work with a 1% sample of the public Twitter stream since it is the only version of the stream that is publicly available. Moreover, the system had to be able to run and handle the incoming load using just a cluster of commodity hardware, namely six desktop computers.

The author of this thesis conceptualized the project and designed the overall system architecture. The implementation effort was aided by a project group of master students under the leadership and supervision of the author within the course of one year.

After giving a general overview of the relevant modules of the system, each of the core modules and the underlying thought process is discussed in more detail in the following sections. Finally, a qualitative analysis of the sentiment classification is provided which solidifies that the best performing classifier that could be determined with the experiments described in the previous chapters yields sufficient performance for real world applications.

6.1 System Overview

A slightly abstracted overview of the modules and the data flow between them is presented in figure 6.1. While the most important modules are described in detail within the following sections, this section will provide a more general overview on how the components work together to achieve the overall goal.

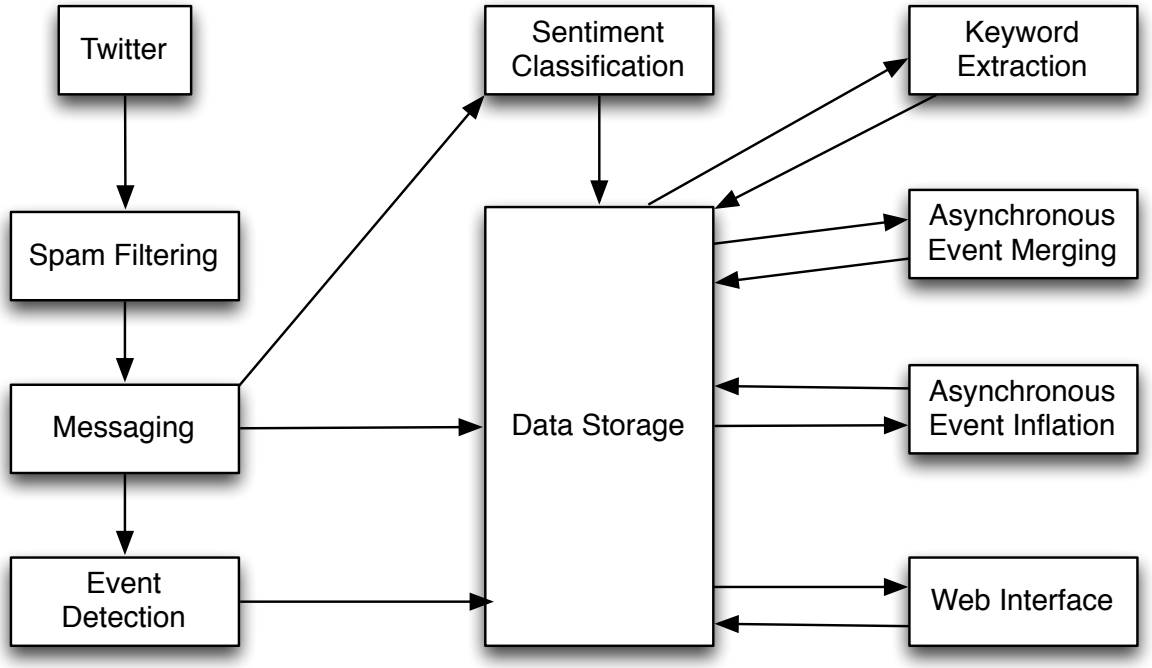


Figure 6.1: Overview of the system architecture.

The starting point is the public Twitter stream where the tweets that are to be analyzed are emitted one at a time in chronological order. In a next step, spam - according to the definition presented in section 5.1 - is filtered out since it is of no interested for further analysis. Non-spam messages are then put into a messaging system[1] from where they can be reliably and asynchronously distributed to all relevant systems through queues to which such systems can subscribe. First, the tweets are directly written to the data storage. The data storage consists of a *Cassandra*[2] database and a *MySQL* database[3]. *Cassandra* was chosen to store the raw tweet data because of the high writing speed and - in anticipation of the large amounts of data - its easy scalability. The application's remaining data such as events, keywords and meta data is inherently rather relational and is stored in the *MySQL* database for ease of accessibility. Second, a service for sentiment classification subscribes to the queue and annotates the tweets with their sentiment. Third, the tweets are handed to the event detection system. Events emitted by the system are written directly to the data storage. Three asynchronous modules are responsible for enhancing the quality and amount of data of already existing events. One module asynchronously merges small events which should actually all belong to one bigger event but get split up in the event detection process. Another module automatically extracts relevant keywords for new or recently changed events. Those keywords can then be used by a third module to asynchronously query the Twitter REST API[4] for additional tweets which did not appear in the restricted public stream. Finally, after all the other processing steps have taken place, the data can be accessed

[1]The messaging system used was *Kestrel*, see `https://github.com/twitter-archive/kestrel`, last checked 28.06.2017.
[2]See `http://cassandra.apache.org`, last checked 28.06.2017.
[3]See `http://www.mysql.com`, last checked 28.06.2017.
[4]See `https://dev.twitter.com/rest/public`, last checked on 25.07.2017.

and analyzed in a web interface.

6.2 Event Detection

The event detection system is based upon a simple *k-means clustering* approach. For the purpose of this application an event E consists of $q > 1, q \in \mathbb{N}$ tweets where q can be chosen according to the applications requirements. Tweets are coming into the system incrementally, one at a time from the public Twitter stream. When a tweet is coming in, it is vectorized to $x \in \mathbb{N}^n$ using a bag-of-words model[5] and its nearest neighbor $y \in \mathbb{N}^n$ is determined, using the cosine distance:

$$dist(x, y) = 1 - \frac{xy}{||x|| \cdot ||y||} \tag{6.1}$$

Once the distance $dist(x, y)$ to the nearest neighbor is computed it is checked whether $dist(x, y) < \rho$ where $\rho \in [0, 1]$ acts as a threshold that has to be manually set. If so, x is added to the same event that y belongs to and if y does not belong to an event yet, a new event is created consisting of x and y. Depending on the value set for q this event is yet just temporary if its size is still below q. Events that do not reach the desired minimum event size in a certain amount of time are discarded and not emitted as recognized events by the system. Moreover, after a certain period of time tweets have to be expired from the pool of the potential nearest neighbors as well to ensure and maintain the system's real-time capabilities.

The crucial point of this approach lies in finding the nearest neighbor efficiently. In the following sections two approaches to achieve this are discussed and compared with one another.

6.2.1 Approximate Nearest Neighbor with Locality Sensitive Hashing

The following section is closely based upon the remarks of Petrović et al. (2010) and McCreadie et al. (2013).

While finding the nearest neighbor to a given data point from a set of previously known data points is well studied, many approaches fail to yield notable improvements over linear search when it comes to high dimensional data such as texts represented by bag-of-word models. A common approach to overcome this obstacles is to relax the problem and be content with an *approximate nearest neighbor* that lies within $(1 + \epsilon) \cdot r$ to the query point, where r is the distance to the nearest neighbor and ϵ has to be determined in a way that it is sufficient for the application at hand.

Indyk and Motwani (1998) proposed *locality sensitive hashing* (LSH) as one of the first approaches that were able to find an approximate nearest neighbor in sublinear time. The basic idea is that data points are hashed into fixed-sized buckets in a way that points within the same bucket have a high probability to be close to one another with

[5] See section 2.1.1 for an introduction.

a given distance measure. Determining the approximate nearest neighbor can then be done by hashing a query point to its corresponding bucket and searching the bucket for nearest neighbor according to the distance measure. Petrović et al. (2010) suggest to use the cosine of the angle between the vectors as distance measure.

Charikar (2002) proposed a hashing scheme for LSH where the probability of the collisions for two points is directly proportional to their cosine distance. The main idea is to divide the vector space into multiple subspaces by intersecting it with k hyperplanes where the number of hyperplanes k can also be interpreted as the number of bits used to represent the bucket keys. A key of length k for a given data point x can then be computed by assigning to a bit $i \in 1, 2, 3, \ldots, k$ the value 0 if $x \cdot u_i < 0$ where u_i is the hyperplane vector of hyperplane i and the value 1 otherwise. Increasing the number of hyperplanes yields a lesser probability of collisions between points which are not close to each other. However, increasing k also lessens the probability of finding the global nearest neighbor. The chance to find the global nearest neighbor can be increased again by creating multiple hash tables that have independently and randomly chosen hyperplanes which generate different subspaces. Given the number of bits k, the probability of collision P^k_{coll} and the probability δ of missing the global nearest neighbor, one can compute the needed number of hash tables as:

$$L = \log_{1-P^k_{coll}} \delta \tag{6.2}$$

In a final step the most probable global nearest neighbor can be chosen from the L potential nearest neighbors by picking the one with the smallest distance to the query point.

Petrović et al. (2010) argue that pure LSH as described above yields poor results for first story detection and also suffers from a lot of variance. The problem seems to be that LSH only is able to reliably find the nearest neighbor of a query point when it is reasonable close to it. However, it may occur that the nearest neighbor is rather far away. Petrović et al. (2010) proposed to not purely rely on LSH but also cache a fixed number (2,000) of recently added documents and linearly search them for a better nearest neighbor. Their results indicate that this error correction strategy works rather well.[6]

Because there is only a finite number of buckets, the number of documents in each bucket will grow without bounds. To prevent this each bucket is set to a fixed size and if the bucket is full the oldest document is expired. However, the document is not globally expired, just in the one overflown bucket in one of the L hash tables. Note that this deprecation strategy may not make sense in all settings but in first story detection in a stream where documents are chronologically ordered it is reasonable. While the previously described strategy ensures a constant number of distance comparisons within the search, the number of comparisons can still be large in practice. Petrović et al. (2010) suggest to only perform $3L$ comparisons. The $3L$ documents are chosen from the set S of all documents that collided with the query document by ordering the elements of S according to the number of hash tables where they collided with the query document.

[6]Please see Petrović et al. (2010) for further details.

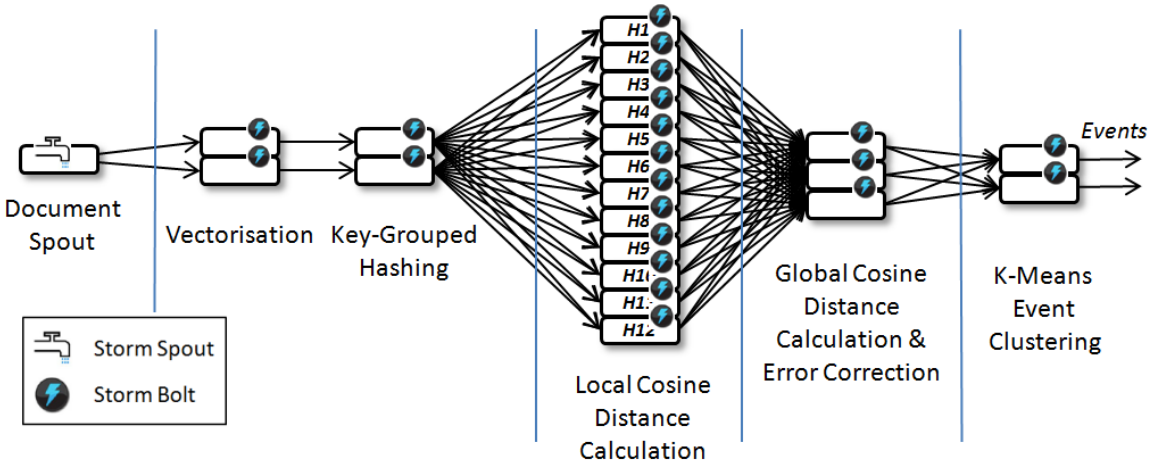


Figure 6.2: Overview of the first story detection storm topology, taken from McCreadie et al. (2013).

McCreadie et al. (2013) introduced a scalable and distributed way to implement the aforementioned LSH based approach for real-time event detection on Twitter. They split up the intermediary steps into smaller blocks that can then be implemented as nodes in a *Storm*[7] topology. This approach results in a robust, scalable and distributed system. An overview of the topology is presented in figure 6.2.

Starting point of the system is the *Document Spout*. It emits the tweets of the stream in chronological order. Note that a spout is an abstract concept, the rest of system does not know whether the tweets are coming directly from the public Twitter stream or are read from a file. Next, the tweets are vectorized in the *Vectorisation* step by representing them as a bag-of-words vector[8]. In the *Key-Grouped Hashing* step, the hash key signature of the tweet for each of the $L = 70$ hash tables is computed using LSH as described above. In the *Local Cosine Distance Calculation* phase, the potential nearest neighbor for each hash table is determined by searching the relevant bucket and the potential nearest neighbor is emitted along with the distance to the query tweet. The bolts in the *Global Cosine Distance Calculation & Error Correction* phase are extracting the global approximate nearest neighbor from the potential results of the previous phase and also apply the error correction step described above. Finally, the clustering of the events takes place in the *K-Means Event Clustering* phase which emits the events according to the general strategy discussed previously in this section.

Note that most of the processing phases can be parallelized very well which makes scaling as easy as adding additional machines to the storm cluster. For example, each incoming tweet can be vectorized in isolation, each of the 70 hash keys for each tweet can be computed in isolation and each bucket can be searched in parallel. By using sufficient amounts of bolts, many tweets can be processed in parallel to drastically reduce the overall processing time.

[7]Storm is a scalable, fault-tolerant and distributed real-time processing framework. Please see Toshniwal et al. (2014) or `http://storm.apache.org`, last checked on 10.07.2017, for an introduction to Storm.

[8]See section 2.1.1 for an introduction.

6.2.2 Approximate Nearest Neighbor with a Fulltext Search Engine

The LSH approach to determining the approximate nearest neighbor that was described in section 6.2.1 involved a lot of manual work with regard to storing and managing the data in an efficient manner. Moreover, it consists of multiple parameters whose values have to be chosen by intuition or have to be systematically investigated in a laborious and time consuming process. To mitigate those drawbacks, it was decided to investigate the feasibility of a well rounded and established system for text data handling: The full text search engine *Apache Lucene*[9].

Lucene's core is a full text index, a data structure for the management and efficient retrieval of text data. Lucene is known to provide reasonable speed, even when working on the local file system, by making use of intelligent batch processing and *memory mapped files*. Documents in the index can be searched for with queries. The query results are enriched with a score that symbolizes the relevance of the results and is based upon tf-idf and cosine distance. However, it should be noted that those scores are always relative and are not invariant to changes in the underlying index. The query most relevant for this work is *MoreLikeThis*[10] which provides documents that are close to the query document with respect to the cosine distance and tf-idf weights for the terms.

Instead of adding all incoming tweets to a hash table like it was done in the LSH approach, they are indexed in a full text index structure that can be searched more easily and directly provides the global nearest neighbor. An overview of the resulting storm topology is shown in figure 6.3. The topology starts with a spout that emits the tweets that are to be processed into the system. In the *Indexing* bolt they are added to the full text index and then are emitted without any further processing. This step cannot be parallelized due to only one process at a time being able to write to a Lucene index. However, while the provided indexing speed is still fast enough for the application at hand it should be noted that this step can also be parallelized by using Solr[11]. The *Distance Computation* step uses the *MoreLikeThis* query to retrieve the nearest neighbor of the query tweet from the index. Additionally, the query tweet and its nearest neighbor are vectorized and their cosine distance is emitted as their distance for later clustering. The result score of the query cannot be used here because it is relative to the query and therefore not applicable for the global clustering with a fixed distance threshold. This step can be parallelized arbitrarily since the index is able to handle parallel queries. Finally, the *Event Clustering* bolt performs the actual event clustering as discussed before and writes the tweets along with their assigned events to the database and updates the index accordingly.

[9] `http://lucene.apache.org`, last checked on 27.06.2017.

[10] See `https://lucene.apache.org/core/5_5_0/queries/index.html?org/apache/lucene/queries/mlt/MoreLikeThis.html`, last checked on 27.06.2017.

[11] Solr is an additional layer of functionality built upon Lucene, see `http://lucene.apache.org/solr/`, last checked 27.06.2017, for further details.

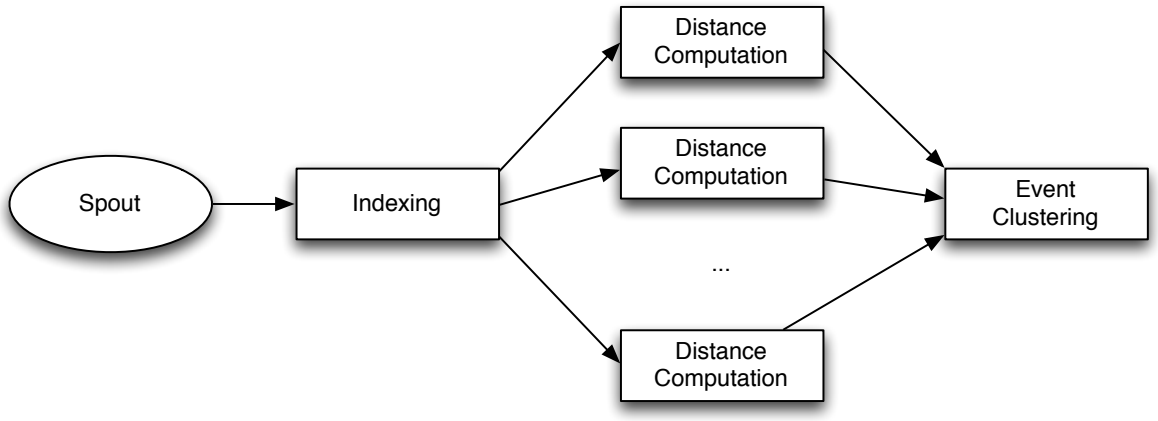


Figure 6.3: Storm topology for real-time event detection based upon approximate nearest neighbor search with Lucene.

6.2.3 Performance Evaluation

To evaluate the performance of the system it was first considered to use the dataset from McCreadie et al. (2013). The dataset consists of 51 million tweets that were collected between June 30th and September 15th of the year 2011. They manually identified 27 events that occurred in the time period and manually searched tweets that were created shortly after the respective event took place. However, only 34,419,701 of the 51 million tweets could be obtained. Many of the missing tweets were part of the manually created events which lead to some events being of very small size, often below the minimum event size of three that was mostly used in this work. Even the bigger events were rather small, most of them consisted only of 50 or less tweets. Since the conditions are not optimal and the results would not be comparable to that of McCreadie et al. (2013) due to the huge amount of missing tweets, it was chosen to create a more suitable dataset and evaluate the systems performance with that.

The event dataset is created from the 33 million tweet corpus introduced in Neubauer (2014). First, a manual internet search was performed to identify events that took place in the relevant time frame of about one year. Events that were not represented in the data - due to the original corpus being collected from the 1% public stream - were later removed which lead to 27 remaining events which are listed in table 6.1. The events were not created by manually extracting tweets such as it was done by McCreadie et al. (2013), but semi automatically. The library Javaluator[12] was used to formulate boolean queries of the form `stormsandy || superstormsandy || hurricanesandy || ((storm || hurricane) && sandy)` to filter the relevant tweets. Such queries assure that only tweets actually containing keywords which are relevant to the respective event are chosen. A purely word based approach analogous to the previous example would yield lots of tweets about storms and hurricanes which are not related to the super storm Sandy.

Evaluation of the system is done by piping the whole corpus of 30 million tweets through the system. The resulting events are then compared to the manually generated events.

[12]For details please see `http://javaluator.sourceforge.net`, last checked on 25.07.2017.

date	event	number of tweets
08.06.12	UEFA Euro 2012	224
17.06.12	election of Mohammed Mursi in Egypt	102
19.07.12	killing spree at Batman premiere in Aurora, Denver	3,856
27.07.12	Olympic summer games in London	3,448
01.08.12	Rover Curiosity landing on Mars	2,839
01.08.12	release of Windows 8	660
25.08.12	death of Neil Armstrong	1,449
25.08.12	Voyager I leaves solar system	77
21.09.12	release of iPhone 5	4,201
12.10.12	EU is awarded the Nobel Peace Prize	169
14.10.12	Austrian Skydiver breaks sonic barrier	1,182
22.10.12	storm Sandy	8,991
06.11.12	reelection of Barack Obama	22,197
18.11.12	Nintendo releases Wii U	366
11.12.12	rocket tests in North Korea	87
14.12.12	killing spree in Sandy Hook School	1,551
21.12.12	Maya prophecy of the end of days	2,517
01.01.13	deaths by fireworks at the Ivory Coast	50
27.01.13	fire in Brazilian nightclub	1,055
10.02.13	Pope Benedict XVI resigns	705
14.02.13	Oscar Pistorius is accused of killing Reeva Steenkamp	831
15.02.13	meteor impact in Chelyabinsk, Russia	484
13.03.13	new pope was chosen	114
07.04.13	death of Margaret Thatcher	31
24.04.13	collapse of textile factory in Bangladesh	52
22.07.13	birth of prince George in Great Britain	1,088
21.08.13	suspected use of chemical weapons in Syria	292

Table 6.1: Manually created events with date and number of tweets in the event.

An event that was created by the system is assumed to be the corresponding event to a manually created event, when the overlap of both is higher than the overlap with all other events emitted by the system. Next, the F_1 score of all the event pairs is computed and the average F_1 score across all events is reported. Since the system was designed to process just around 50 tweets per second the evaluation process took about 10 days for each full run. Hence, the parameter tuning was partly done with informal experimentation on a subset of the data between the full runs that is not documented here. The results of the complete evaluation runs are presented in table 6.2. During the first run of the LSH approach it was noted that there were quite some difficulties in the implementation that were not mentioned by McCreadie et al. (2013). The memory demands of the system were more than the available experiment hardware could bear which often led to out-of-memory errors. Moreover, the throughput was much lower than expected and in some cases did not reach the necessary 50 tweets per second. Also, there were difficulties with the internal storm mechanism of field grouping that failed to efficiently distribute some parts of the computation. Due to this difficulties which did not arise with the Lucene approach, the LSH approach was discarded after the first

NN	min event size	dist thresh.	avg recall	avg prec	avg F_1
LSH	3	0.45	0.03630	0.92095	0.06724
Lucene	3	0.45	0.01073	0.26857	0.01856
Lucene	4	0.65	0.02550	0.54105	0.04376
Lucene	10	0.55	0.00474	0.10817	0.00886
Lucene	3	0.60	0.07301	0.66642	0.12044

Table 6.2: Results of the event detection evaluation. NN stands for the methodology to find the nearest neighbor, the other column labels are self explanatory.

full evaluation.

While the initial approach based on Lucene performed worse than the LSH approach, some parameter tuning[13] led to an average F_1 score that almost doubled the LSH performance. It should also be noted that increasing the minimum event size yields significant drops in performance and should not be applied. The overall best score $F_1 = 0.12044$ seems rather low on first sight. However, due to the nature of the semi automatically created dataset the original events are not even 100% clean to begin with. Moreover, it was noted that the system tends to generate multiple small events which should instead all be pooled into one big event, a solution to this problem is discussed in the following section. While the inter event performance seems to be not optimal, it should be noted that 26 of 27 original events were recognized by the system which is a very good result. Through additional qualitative, informal evaluation on the public Twitter stream the final configuration with a minimum event size of three and a distance threshold of 0.60 was deemed to work sufficiently well for the requirements of this project.

6.2.4 Fixing Event Size Issues with Asynchronous Merging and Inflation

Looking at the events generated by the best parameter configuration presented in section 6.2.3 it was noted that the events were rather small and still were missing lots of tweets. This result can be explained with the observation that the original events are often split up by the system and are recognized as multiple smaller events.

A solution to this problem is to merge the small events to a single big event. This is done asynchronously and independently to the event detection topology. Additionally to storing the tweets in the Lucene index and the data base, the index also maintains a document for each event whose text is the concatenation of all associated tweets. Periodically, a process loads unmerged events and uses a *MoreLikeThis* query on the event document index to identify similar events. The event with the highest result score is chosen and the cosine distance between the result and the query event is computed. When the distance reaches a manually determined threshold t, the events are merged by adding the smaller one to the bigger one and updating the database and the index accordingly. For merge attempts where the distance is still below $2t$, the events are marked as potential merge and have to be manually approved in the web interface.

[13]Note that there were additional implementation details which are not mentioned here.

Merge attempts with larger distances are not considered to be sufficiently similar for merging.

While merging existing events enhances the analysis capabilities for the user greatly, one still has the feeling that a lot of information is missing. Consider for example an event that was found by system after the new trailer of the TV series *Game of Thrones* was released. After applying the event merging, the event still just consisted of a few dozen tweets. This seems unreasonable because *Game of Thrones* was one of the most popular TV shows of the time. However, when only receiving a random sample of 1% of all tweets from the live stream, a lot of information is missed. To remedy this shortcoming another asynchronous system was created. The system periodically loads events that are newer than one week[14] from the database, reads the event's keywords and performs a query towards the REST API of Twitter. Through this API it is possible to somehow mitigate the 1% restrtction by providing a filter for the tweets one is interested in. The more specific the filter, the less unrelated data is provided and often the limitation is not even reached while gaining all relevant tweets for an event. For filtering, the automatically extracted keywords of the events where used as query terms. The resulting tweets are put through the topology the same spout that emits the streaming data and with the previously discussed event merging mechanism the tweets are assigned to the relevant events in the long run. Using this technique the *Game of Thrones* trailer event could be increased in size from a few dozens to multiple thousands. The results were similar for multiple other events that haven qualitatively investigated in that manner. Simulating this process would be very complex and full of assumptions for unknown parameters that the approach was deemed feasible for the application based upon the qualitative results and was not investigated more systematically.

6.3 Keyword Extraction

Relevant keywords for each of the recognized events are generated for multiple purposes. First, they are required for the asynchronous event inflation system described in section 6.2.4. Second, they are to be presented to the user in the web interface as a summary of the event. Third, they are used for filtering tweets of events with respect to the respective sub topic each keyword stands for.

In the following sections two approaches for keyword extraction of events are presented and a qualitative comparison between them is discussed.

6.3.1 Hybrid TF-IDF

Simsek and Karagoz (2014) found a common *TF-IDF* approach with a modification that was proposed by Inouye and Kalita (2011) to be sufficient for a well performing keyword extraction from a set of tweets. They also presented an approach which considered the sentiment of the tweets to boost positive keywords and punish negative ones but since that is not the goal here, the step is omitted.

[14] One cannot query farther back because of limitations in the Twitter API.

The basic idea of *TF-IDF* is that a word is important to a document when it occurs frequently in this document but does not occur frequently in the general collection of all documents. Since tweets are not traditional documents due to their short length, a hybrid definition is needed for the *TF-IDF* to work well (Inouye and Kalita 2011). For computing the term frequency *TF* all tweets are concatenated to a single document while computation of the *IDF* part treats each tweet as a separate document.

With the aforementioned definitions the weight $W(w_i)$ of a word w_i can be computed as follows:

$$W(w_i) = tf(w_i) * log_2(idf(w_i)) \tag{6.3}$$

$$tf(w_i) = \frac{\#\text{OccurrencesOfWordInAllTweetsOfTheEvent}}{\#\text{WordsInAllTweetsOfTheEvent}} \tag{6.4}$$

$$idf(w_i) = \frac{\#\text{Tweets}}{\#\text{TweetsInWhichWordOccurs}} \tag{6.5}$$

Because the values of *TF-IDF* are often not well interpretable, the results are normalized by assigning the keyword with the highest weight the value one and scale the other weights accordingly. The ten keywords with the highest weights are then chosen as keywords for the event.

6.3.2 PageRank

In contrast to the *TF-IDF* approach, the *PageRank* approach does not purely rely on the term frequencies of the words but builds a graph structure which also takes the relative position of the words to one another into account. While very sophisticated methods for keyword extraction based upon graphs have been proposed, for example by Abilhoa and De Castro (2014), they are computationally rather intensive and were deemed not feasible in the real-time processing scenario at hand. The approach investigated here is based upon the *PageRank* (Page et al. 1999) algorithm. The basic idea is that nodes in a graph which are linking to another node with a directed edge emphasize the importance of the other nodes. With an iterative procedure that finally converges, one can use this algorithm to asses the importance of all nodes based upon their references from other nodes[15] which makes it suitable for keyword extraction.

The remaining question is how to build the initial graph. Before building up the graph the text data is filtered by extracting nouns and verbs, removing punctuation and lemmatizing the remaining tokens. Abilhoa and De Castro (2014) suggest to initialize the graph with the remaining tokens as nodes. A directed edge is added from a node A to a node B when B occurs later in the text than A and the distance - counted by the words between them - is not higher than a fixed threshold.

[15]See Page et al. (1999) for further details.

event name	event description	#tweets
iPhone 5	Apple introduced the iPhone 5 which had a notably bigger screen than its predecessors	2,502
Curiosity	The robot Curiosity landed on Mars in August 2012	1,137
Egypt Election	Mohamed Morsi was elected in Egypt in 2012	60
Olympics 2012	2012 Olympics in London.	2,146
North Korea Rocket	North Korea placed a satellite in space using a rocket	69
U.S. Election	The U.S. election in 2012 with the candidates Barack Obama and Mitt Romney	11,671

Table 6.3: Overview of the events that were used for evaluation.

Such a graph is created for a document whose content is the concatenation of all tweets in the event. The weights are all initialized with the same starting value and the *PageRank* algorithm is run on the graph until it converges. The weights generated for each token are normalized relatively to the highest weight and the top 10 tokens are emitted as the most relevant keywords for the event.

6.3.3 Comparison of the Approaches

To compare the two approaches it was chosen to qualitatively evaluate the extracted keywords for a few semi automatically created events. The events are an intermediate subset that was extracted from the event dataset introduced in section 6.2.3 while it was still in the process of being constructed and consisted of the events that are outlined in table 6.3.

The actual comparison is done by generating keywords with both approaches and qualitatively evaluating the results. The results are presented in tables 6.4 to 6.9.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
iphone 5	1.0	iphone 5	1.0
retweet	0.1289	apple	0.1664
apple	0.1116	iphone	0.1455
follow	0.0896	phone	0.1399
phone	0.0878	release	0.0684
win	0.07802	time	0.0532
iphone	0.0744	case	0.052
people	0.0659	screen	0.0417
brand	0.0615	contract	0.0404
today	0.0532	ipad	0.0368

Table 6.4: Extracted keywords for the iPhone 5 event.

Table 6.4 depicts the extracted keywords for the *iPhone 5* event. A notable observation is the ranking of the keyword "retweet" as second most important by the *TF-IDF* approach. Obviously, the word is completely irrelevant for the introduction of the iPhone 5 but seems to have been prevalent in many tweets. Moreover, one of the most contentious changes to the iPhone design - the big screen - is not reflected in the *TF-IDF* list. Apart from the keyword "time" there are no irrelevant terms in the keyword set of *PageRank* and the term "screen" is also included. Hence, *PageRank* seems to have extracted better keywords for this event.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
mars	1.0	mars	1.0
curiosity	0.5135	curiosity	0.7501
rover	0.4787	rover	0.7227
msl	0.2531	nasa	0.3651
nasa	0.1845	msl	0.1392
landing	0.1686	planet	0.0883
crater	0.1585	science	0.0827
gale	0.1546	space	0.0725
back	0.1053	mission	0.0718
surface	0.1029	martian	0.0668

Table 6.5: Extracted keywords for the curiosity event.

Results for the *Curiosity* event are shown in table 6.5. Both approaches manage to extract the most relevant keywords in the top 5 of the keyword list. While *TF-IDF* also includes additional information in form of the rover's landing site "gale crater" it also includes the irrelevant word "back". There is no clear winner to be determined here and a preference towards one of the approaches would be rather subjective to the overall system requirements.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
egypt	1.0	egypt	1.0
vote	0.7246	vote	0.2928
morsi	0.5072	morsi	0.2262
voted	0.2609	election	0.1661
elected	0.2319	referendum	0.1434
egypt's	0.2319	president	0.1082
president	0.2029	talks	0.1029
election	0.2029	round	0.1006
opposition	0.1449	army	0.0957
egyptians	0.1449	unity	0.0909

Table 6.6: Extracted keywords for the egypt election event

The keywords of the event *Egypt Election* are presented in table 6.6. Again, the top keywords are the most relevant ones and have been detected by both algorithms. However, the very relevant keyword "election" is ranked rather low for *TF-IDF*. The lower ranking keywords of both approaches are all relevant to the event. Which approach is better can not objectively be determined in this case.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
london	1.0	london	1.0
olmypics	0.7729	olympics	0.8643
olmypic	0.523	olympic	0.3858
london 2012	0.275	london 2012	0.2471
games	0.1878	games	0.1463
amazing	0.1055	ceremony	0.1145
ceremony	0.0991	team	0.0702
opening	0.0939	time	0.0594
sad	0.0767	summer	0.0567
weeks	0.0733	medal	0.056

Table 6.7: Extracted keywords for the Olympics 2012 event.

Looking at the keywords for the *Olympics 2012* event which are presented in table 6.8 one notices again that both approaches were able to identify the most important keywords of the event. However, the *TF-IDF* approach again identifiers irrelevant words such as "amazing", "sad" and "weeks" as relevant keywords, while *PageRank* only emitted one irrelevant keyword ("time"). Hence, *PageRank* was considered to have a slight edge for this event.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
korea	1.0	korea	1.0
north	0.9608	rocket	0.9395
rocket	0.8824	north	0.8994
launch	0.7549	launch	0.5875
long	0.3824	missile	0.1685
range	0.353	month	0.1002
launched	0.1471	suspected	0.0987
missile	0.1275	korean	0.098
month	0.0882	dec 22	0.0935
korean	0.0882	technology	0.0887

Table 6.8: Extracted keywords for the North Korea rocket event.

The keywords for the event of the *Korean rocket launch* are shown in table 6.8. Both approaches included the irrelevant word "month" into the keyword list. While *PageRank*

included another rather irrelevant term ("technology") the rest of the keyword list of *TF-IDF* seems reasonable. Hence, *TF-IDF* has a slight edge in this case.

TF-IDF		**PageRank**	
keyword	**weight**	**keywords**	**weight**
obama	1.0	obama	1.0
vote	0.9912	romney	0.6382
romney	0.7215	election	0.4578
election	0.4007	vote	0.2878
mitt	0.2295	mitt	0.1455
elected	0.2046	president	0.1414
president	0.1597	tcot	0.0673
voted	0.1492	ryan	0.06687
people	0.1117	years	0.0611
black	0.0935	time	0.0599

Table 6.9: Extracted keywords for the U.S. Election event.

Finally, the last investigated set of keywords - the 2012 U.S. Presidential election - is presented in table 6.9. While *PageRank* includes multiple irrelevant words it is still considered to have the upper hand in this case. When looking at the most important keywords it seems reasonable to rank the names of the candidates at the top, followed by "election". *TF-IDF* ranks the term "vote" before the name of one of the candidates which is completely undesirable. Hence, *PageRank* is considered to perform better in this case.

Both of the investigated approaches yielded reasonable results that can be useful. However, due to *PageRank* having a slight edge it was chosen to use this for the final version of the system.

6.4 A Web Interface for Explorative Data Analysis

One of the requirements of the system was the capability to exploratively analyze the detected events in a modern web interface. The most relevant parts of this web interface are presented and discussed in this section along with a qualitative assertion of the general purpose sentiment analysis results.

6.4.1 Event Overview

Events that were detected by the system can be analyzed in the web application. A screen shot of an example event is presented in figure 6.4. The event shown here was detected in early March 2016 and consists of messages about new trailer to the popular TV series *Game of Thrones*.

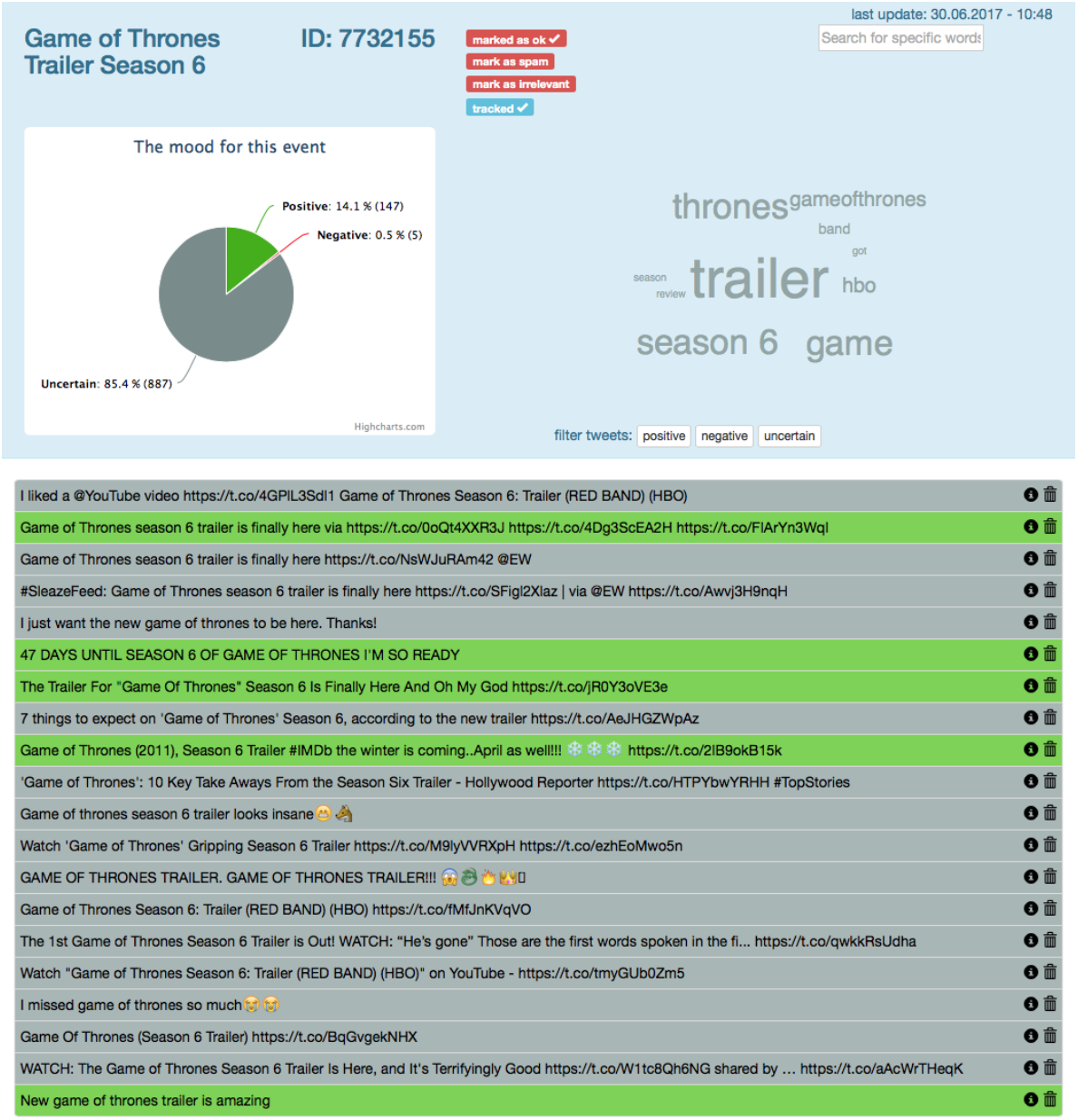


Figure 6.4: Screen shot of the event overview page from the web interface.

In upper left corner, the title of the event is displayed, in this case "Game of Thrones Trailer Season 6". A title is assigned automatically by concatenating the most relevant keywords of the event. Since those titles are often not easy to read, a title can also be manually assigned as it was the case for this example event.

Next to the title, the labeling panel can be seen. An event can be labelled as either *ok*, *spam* or *irrelevant*. The label *ok* means the event is an event the user is interested in and such events are displayed prominently in the event list for easy access. Even though spam filtering is applied to the input stream of the system, some spam tweets sometimes make it through. Hence, it makes sense to be able to label events containing mainly spam as *spam* to make use of them later for refining the spam classification algorithm. Another class of events is *irrelevant*. Often generic events with topics such as "love",

"eating spaghetti" or "sleeping" are detected when people report their daily activities. Since those events are not of general interest, the system is capable of annotating them with a label to later be able to filter out those events if such a behaviour is desired. Additional labels could be easily added when the need arises. Moreover, an event can be marked as *tracked* which means the asynchronous event inflation process will be activated for this event and load additional messages in the background based upon the keywords.[16]

The tweets the event consists of are presented in a tabular view at the bottom. The background color indicates the sentiment the classification system assigned to the tweets. Classification was performed with the best performing classifier that was found in the previous experiments.[17] Tweets that have been assigned to the event but not actually belong to it can be manually removed by clicking the garbage can on the right of each tweet. Clicking the info button next to the garbage button opens a dialog that is presenting additional data to the respective tweet, such as the author or time and location of posting. Moreover, the detail dialog enables the user to directly interact with the tweet in form of responding, retweeting or saving the tweet as a favorite.

Below the title there is the sentiment overview in the form of a pie chart depicting the overall sentiment distribution of the event.

A tag cloud is presented next to the pie chart. This tag cloud is made up of the most relevant keywords. The size of a word indicates the relevance score of the keyword and the text color indicates the sentiment towards the word based upon the ratio of the sentiment label of the tweets in which the word occurred. Since most of the tweets in this event are of *uncertain* sentiment, all the keywords are written in grey. This visualization enables a user to instantly perceive what are the most important terms of the event and which sentiment is expressed towards them. Moreover, the tags are clickable to filter the tweets in the tweet box below to only display messages that contain the tag that was clicked.

Filtering the displayed tweets according to their sentiment can be done with the three buttons below the tag cloud. This feature can be beneficial if one quickly wants to assess which aspects of the event are perceived as positive, negative or uncertain.

For various events it may be interesting to see how exactly the sentiment developed over time through the course of the event. Hence, the graphs presented in figure 6.5 are shown below the tweet table. The upper graph displays the absolute volume of positive and negative tweets where the positive count is shown above zero and the negative count is shown below zero. Moving the cursor over a data point displays the absolute number of messages. The lower graph shows a mood score which combines the positive and negative values to a single score for easier visualization of the ratio:

$$mood\_score = \frac{\#\text{positive} - \#\text{negative}}{\#\text{positive} + \#\text{negative}} \tag{6.6}$$

Uncertain tweets are not considered for these graphs.

[16] See section 6.2.4 for further details.
[17] See chapter 5 for further details.

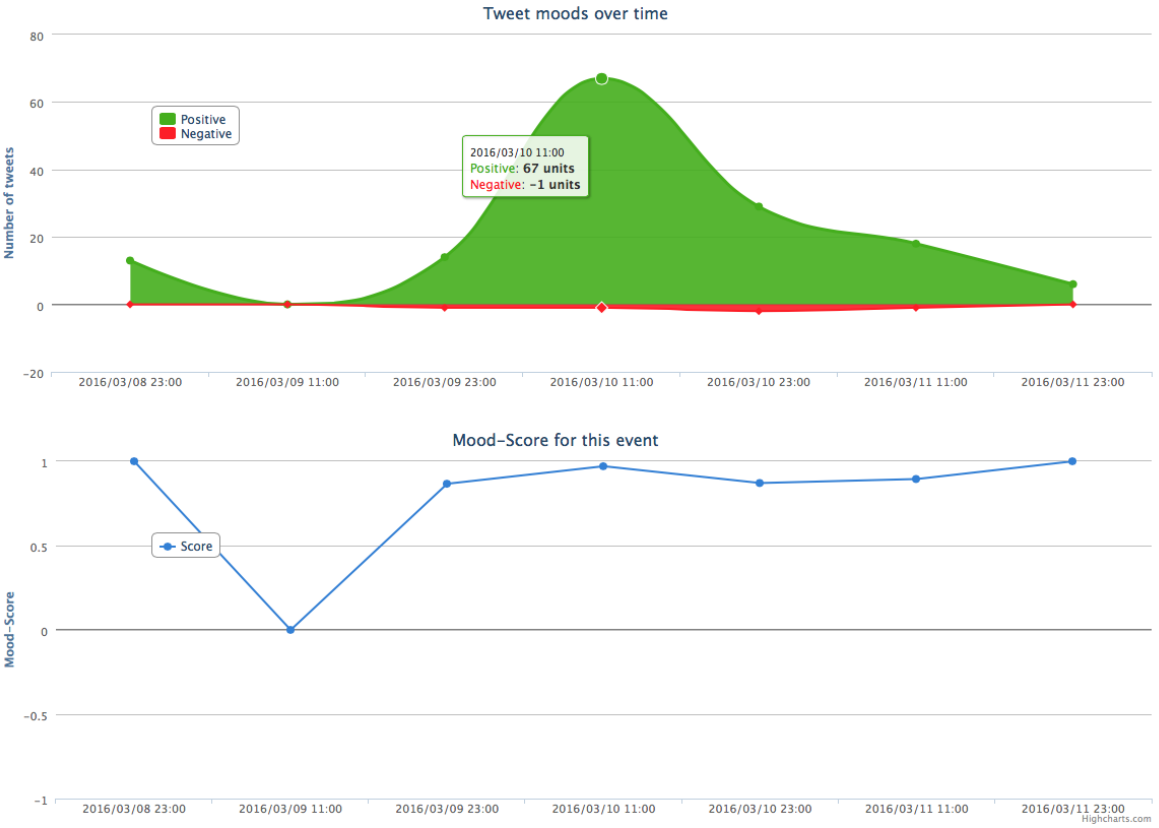


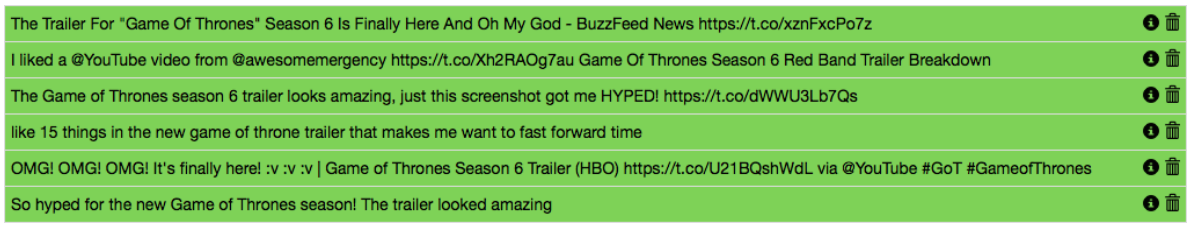
Figure 6.5: Screenshot of the sentiment timeline view from the event overview.

Moreover, it can be relevant to be able to explore a subset of the event by filtering the analyzed tweets with a full-text search. In the top right corner there is a text field where one can enter the search terms. Only tweets matching the search are displayed and all graphs are updated accordingly. In this way the user can dive deeper into sub aspects of the event by iteratively refining the search.

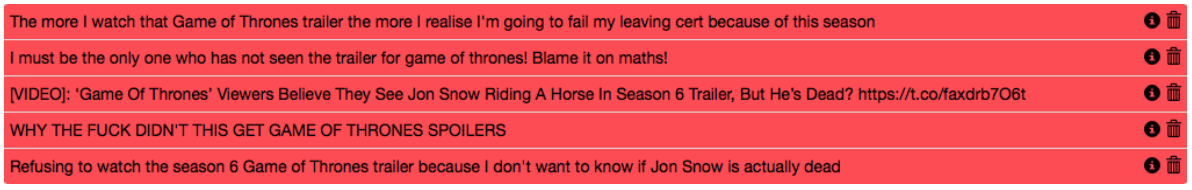
6.4.2 Qualitative Assessment of the Sentiment Classification

For assessing the viability of the general purpose sentiment approach, some events are presented and discussed in this section. This assessment is done using unlabeled data from the live system and, hence, can only be of qualitative nature.

A subset of tweets from the Game of Thrones trailer event introduced in the previous section is presented in figure 6.6. Looking at the examples it can be said that the classification seems to work reasonably well. Tweets classified as *positive* consist mainly of hype for the trailer and the new season. The negative messages mostly express regret of not having seen the trailer yet or the TV series taking so much time so other real live activities will be neglected. Examples of the *uncertain* class mostly consist of rather objective statements that the trailer came out. One exception being the first message which could be argued to be of rather positive sentiment since it is stated that the author liked the trailer on Youtube. While not being perfect, the classification seems to work as intended in a real world scenario.



(a) Positive tweets



(b) Negative tweets

(c) Uncertain tweets

Figure 6.6: Tweets of the Game of Thrones trailer event for qualitative assessment.

For further qualitative assessment, the pie charts of some exemplary events that were detected by the system are shown in figure 6.7.

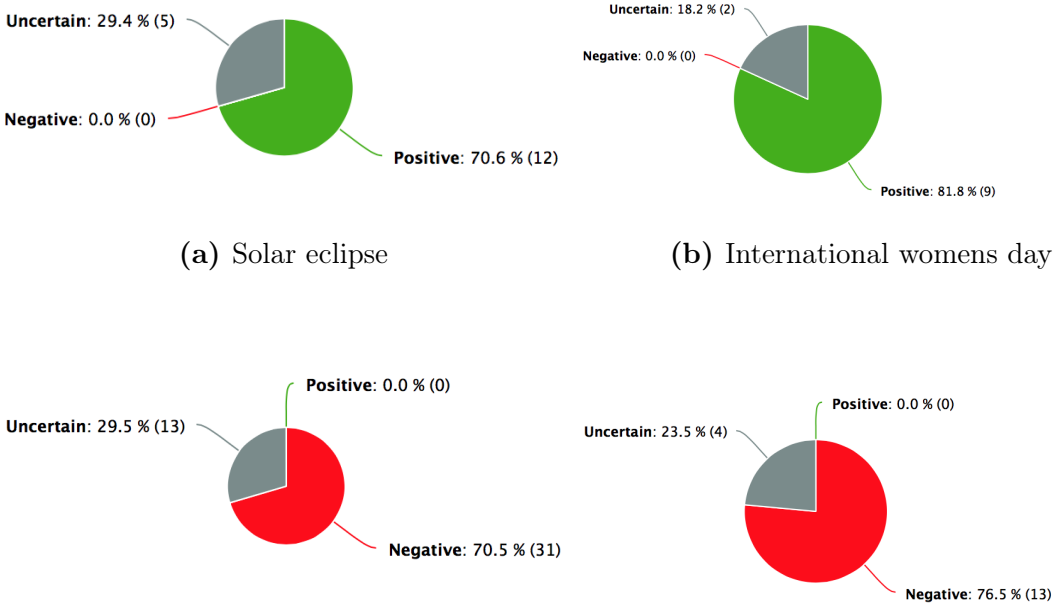
Figure 6.7a presents the chart of an event about a solar eclipse in 2016. Inherently to the event, the sentiment is mostly *positive* with some rather objective statements that were deemed *uncertain* by the system. These results are in line with the public reception of such an event, most people would probably classify a solar eclipse as an interesting and positive event.

Another rather positive event is presented in figure 6.7b: The international women's day. Similar to the solar eclipse, this event is of inherent positive nature and the tweets about it consist mostly of people wishing others a happy international womens day.

The event whose sentiment is depicted in figure 6.7c was detected by the system shortly after Donald Trump had to cancel one of his campaign events in Chicago due to chaotic protests of opposed citizens, mostly students. Chaotic protests are in general not a positive thing which is matched by the overall sentiment of the event. Not even a single positive tweet was recorded.

In 2016, professional Golf player Jim Furyk had to undergo complicated hand surgery and it was not clear at the time if he could ever play golf again. The pie chart of the corresponding event is presented in figure 6.7d. Naturally, many of his fans tweeted their regret about his injury and their worries about him not returning to the game. The sentiment classification of the event reflects this well.

All in all it can be said that the qualitative assessment indicated that the proposed method for general purpose Twitter sentiment analysis is viable in a real world scenario



(a) Solar eclipse **(b)** International womens day

(c) Cancelled Trump event in Chicago **(d)** Jim Furyk Surgery

Figure 6.7: Sentiment pie charts of some detected events.

and produces feasible results from which relevant insights about current topics and the sentiment toward them can be gained.

6.4.3 Location Based Event Analysis: Super Bowl

The general motivation of the application presented in this section is that many events can have an association to certain locations and the sentiment of the tweet's authors may be related to the location they tweeted from. To be able to discover such coherences, a web application with interactive data exploration capabilities over a period of time has been developed. While the standalone application discussed in this section is decoupled from the event detection system it would easily be possible to import a detected event from the other platform.

Since this application also is a proof-of-concept that relations between an event, user sentiment and location can be extracted from the public Twitter stream, the application is so far restricted to the USA and some parts are still hard coded for the analysis of example event. However, all the specific parts could be easily generalized with some effort that, unfortunately, was not possible to do with the resources available.

A screen shot of the application's main view is shown in figure 6.8. The event that is analysed here is the final game of the Super Bowl XLIX between the *Seattle Seahawks* and the *New England Patriots*.

On the top left corner the map of the U.S.A. is presented. The colored circles represent aggregated clusters of tweets which were tweeted near the location of the circles center.

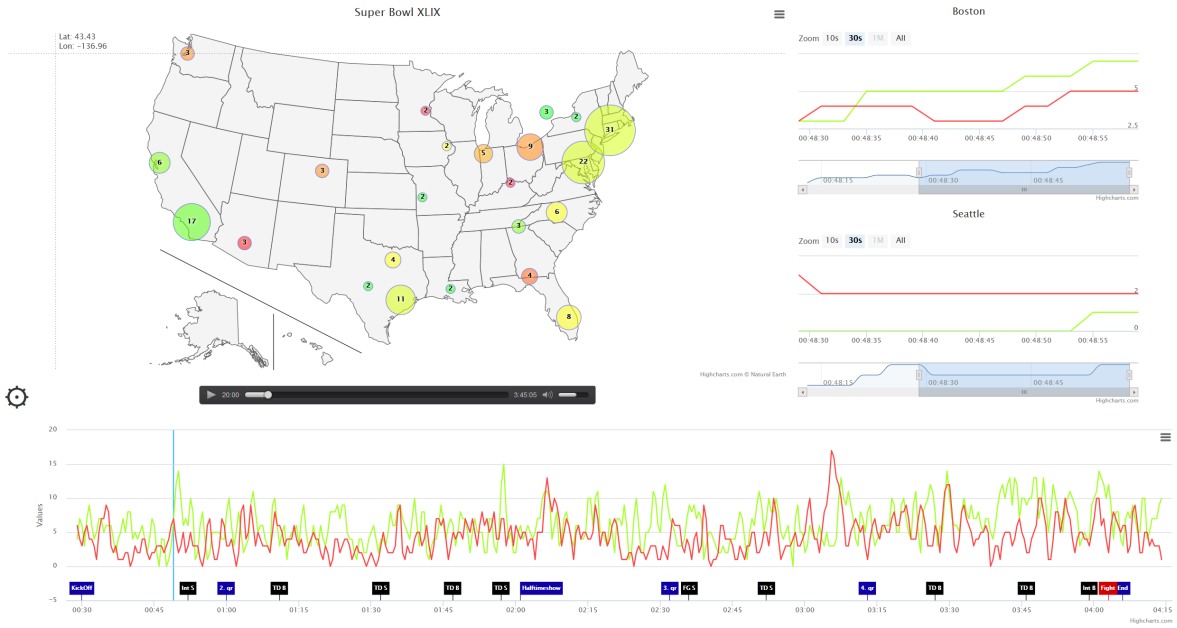


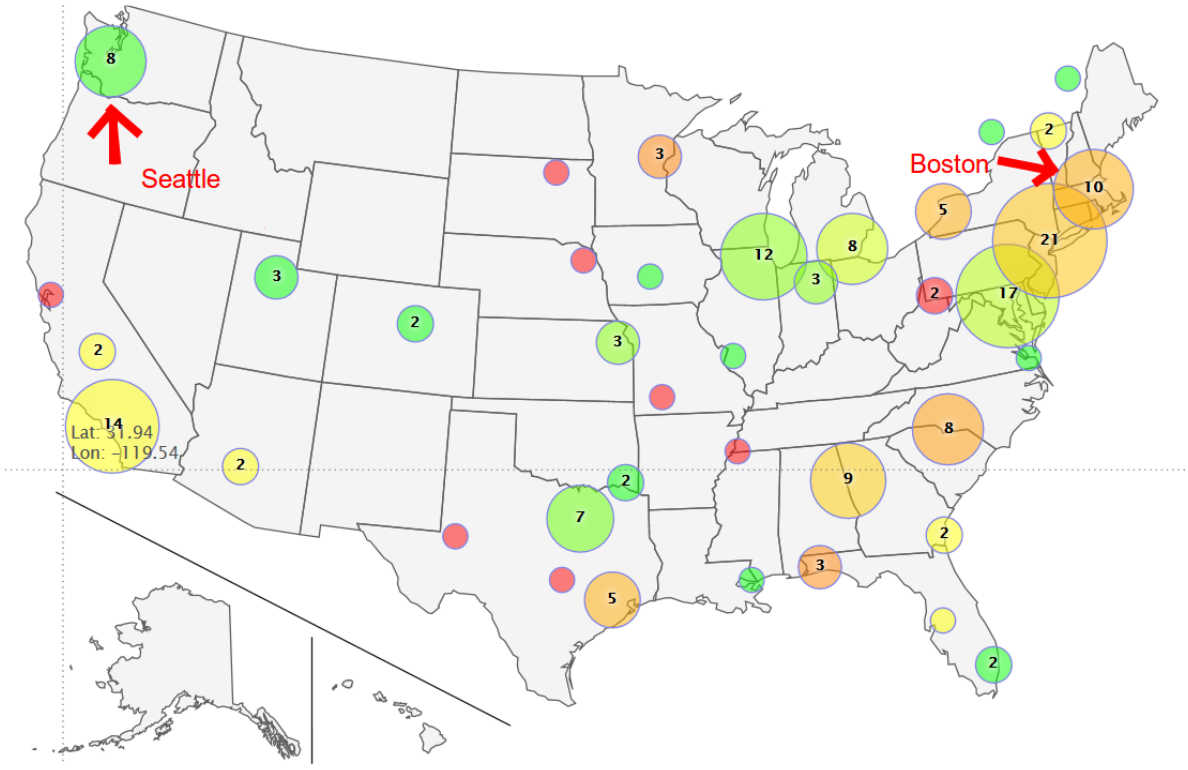
Figure 6.8: Screenshot of the Super Bowl application main view.

Bigger circles represent bigger events. The color of a circle denotes the sentiment where it would be green for only positive tweets, red for only negative and yellow for an even number of positive and negative tweets. The actual color is an interpolation between the aforementioned colors based upon the ratio of positive and negative tweets. Below the map there is a control panel similar to that of a video player. When hitting play, an audio report of the Super Bowl final game is played and the data on the map is animated accordingly to reflect the tweets that were posted at the time.

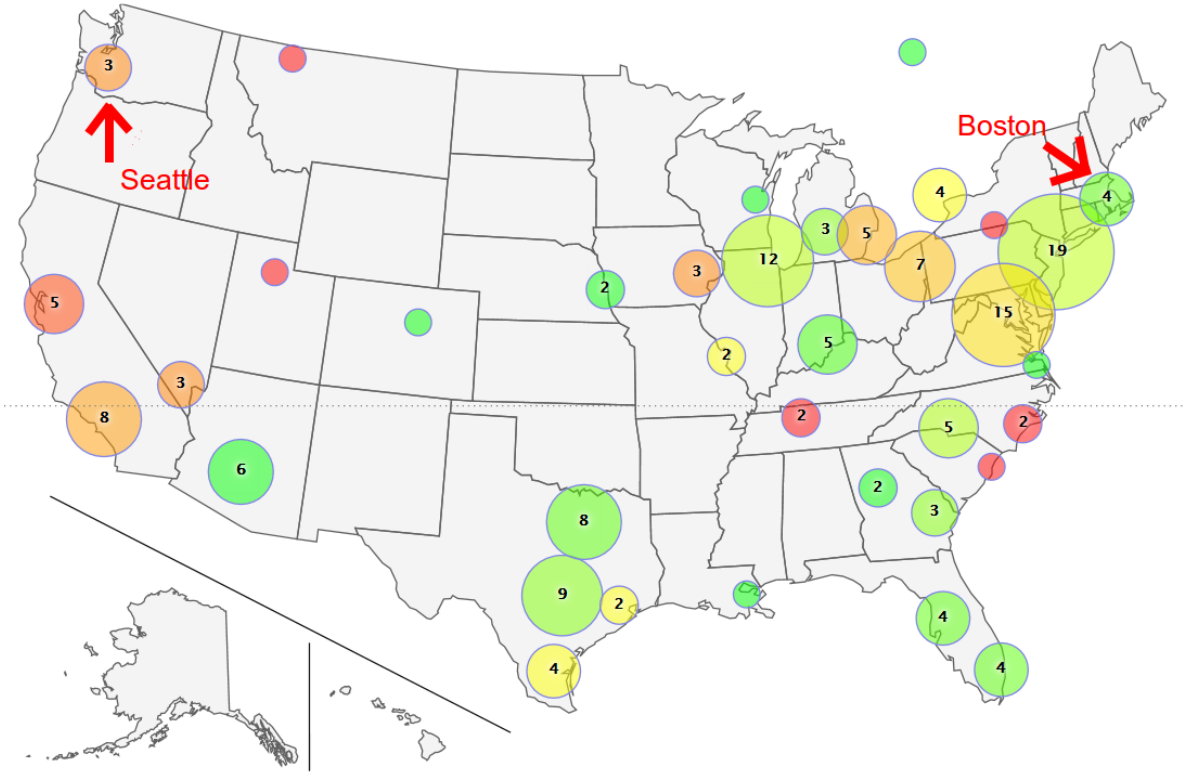
The graph on the bottom is a visualization of the volumes of all positive and negative tweets that are related to the super bowl event. This graph can be used select only a certain time interval by dragging a window on top of it. Moreover, it is annotated with the most relevant events of the game, such as field goals or touchdowns. Similar graphs are shown in the top right corner but those only visualize the tweet volume of the home cities of both teams, Boston and Seattle.

In figure 6.9a a screen shot of the map is presented which depicts a situation in the game where the Seattle Seahawks were about to score a touchdown. Even though there are very few tweets available, the sentiment in Seattle was strictly positive, while the sentiment in Boston was rather negative. A screen shot around the time Boston scored a touchdown is shown in figure 6.9b. Due to the small amount of tweets this could only be seen as an indication. Unfortunately, only about 1-3% of tweets are annotated with geo tags and not all relevant tweets for the time frame could be obtained because of restrictions in the public Twitter API usage.

In summary, it can be said that there is an indication for a relationship between what is tweeted and events in the real world where the sentiment can be tied to certain locations. While the association between the location and the sentiment in the Super Bowl example was obvious and known before hand, the result that the expected behaviour is actually reflected in the data opens up new possibilities for characterizing events based upon



(a) Screenshot of the Super Bowl analysis application around the time the Seattle Seahawks scored a touchdown.



(b) Screenshot of the Super Bowl analysis application around the time the New England Patriots (Boston) scored a touchdown.

Figure 6.9: Screenshots of touchdown events.

the local sentiment of the users. However, reliable analysis results are relatively hard to obtain due to the lack of a geo tag for most tweets. Even for a popular event such as the Super Bowl, there is just very little data available.

6.5 Summary and Conclusion

A distributed architecture for real-time event detection with explorative data analysis capabilities has been conceptualized and successfully implemented. Multiple approaches for the various sub problems were investigated and the respective best solution was applied successfully to the application. An approach based upon k-means in conjunction with a full-text index for finding the nearest neighbor proved to be a viable system for detecting the events. Keywords were extracted with an approach based upon a graph representation and the PageRank algorithm. Using the previously determined keywords, the system is able to acquire additional information for the events from the asynchronous REST API of Twitter to somewhat mitigate the drawbacks of using the 1% sample stream.

Only the publicly available Twitter data was used by the system and processing was done on commodity hardware, namely six desktop computers. Due to the strict decoupling of the components, the system is easily scalable by adding additional hardware since most of the data processing and analysis can be parallelized.

The system was able to detect events in real-time and provide useful information about them. Moreover, it could be shown that the sentiment detected within an event is in some cases tied to the user's location which can provide useful insights. Finally, the resulting web interface could be used to verify that the best classification algorithm for general purpose Twitter sentiment analysis that was determined in previous experiments yields viable and reasonable results in a real world scenario.

7 Summary and Conclusions

After the detailed review of the state-of-the-art methods, it was clear that many methods rely on manual feature engineering coupled with various preprocessing methods. However, the process of choosing the combinations of such methods was seldom discussed in detail. Hence, a large scale study was performed with regard to the efficiency of combining preprocessing pipelines with various feature extraction methods. It revealed that commonly used techniques are actually harmful for the performance in many cases. Moreover, it was indicated that using the same preprocessing pipeline - as is commonly done in state-of-the-art methods - for all feature extraction methods is not optimal and that different preprocessing techniques have to be applied for different feature extraction methods to yield maximum performance. Those insights are not only relevant for reliable general purpose Twitter sentiment analysis but directly apply to the related fields as well.

Next, a high quality dataset was introduced that could be used to evaluate algorithms for their feasibility to perform reliable general purpose Twitter sentiment analysis. It was chosen to mainly investigate two archetypical approaches, manual feature engineering in combination with various classification algorithms and deep convolutional neural networks where no feature engineering is necessary. Moreover, a rather novel idea of confidence based rejection that only made use of two of the three sentiment classes in the training process was investigated for both archetypes, mainly with the purpose of harnessing the related domain data efficiently. All of the investigated methods yielded feasible performance, while the lead was taken by a manual feature engineering approach. Furthermore, indications where found that automatically generated resources such as lexicons can yield better performance when regenerated from corpora which are closer related to the domain at hand. It was also investigated how active learning could be harnessed to remedy the necessity of labeling huge amounts of data for the neural network training by sophisticatedly choosing samples to annotate. Results indicate that a system trained with the active learning approach is able to achieve similar results to a system that was trained with much larger amounts of randomly selected samples. Moreover, the best neural network that was found was trained with the tweets that were annotated within the course of the active learning experiments. To the best of the author's knowledge, there is no other work going in this direction in the related domains of Twitter sentiment analysis and, hence, they may also benefit from the new insights. Nevertheless, the best performing neural network that was found still performed sub par to the best system based on manual feature engineering. However, it should be stated that the archetypical network architecture investigated in this thesis often is only one part of the more complex approaches. Hence, future work in this field should not generally discard deep convolutional neural networks as a whole for reliable general purpose Twitter sentiment analysis but probably investigate more complex architectures.

Finally, a proof-of-concept architecture of a real-time event detection system with explorative data analysis capabilities was introduced and implemented successfully. The system runs on commodity hardware and only uses the publicly available 1% sample stream from Twitter. Even with the aforementioned constraints the system was able to detect relevant events based upon the available data. The events can be comfortably analyzed in a modern web interface. Performing a qualitative evaluation on previously unlabeled real world data indicated that the best classification algorithm that could be determined in the previous experiments yields sufficient performance and plausible results for the application at hand. Future work may consist of investigating further strategies for some of the partial problems that could not be evaluated in full detail within the scope of this work. However, a well rounded starting point for further research in the direction of real-time event detection on the public Twitter stream could be provided.

Furthermore, the experimental results solidified the initial hypothesis that data from the related domains is undoubtedly not representative for the public Twitter stream and, indeed, does not match the problem of reliable general purpose Twitter sentiment analysis as closely as one might think at first glance. Hence, reliable general purpose Twitter sentiment analysis should be certainly considered as a new and delimited but also closely related field from which various applications could benefit greatly.

The work presented in this thesis laid down the foundation for the novel field of reliable general purpose Twitter sentiment analysis by surveying state-of-the-art methods for related problems, investigating the transferability of the archetypical approaches and datasets to the new problem and verifying the feasibility of the results in a real-world scenario.

Bibliography

W. D. Abilhoa and L. N. De Castro. A Keyword Extraction Method from Twitter Messages Represented as Graphs. *Applied Mathematics and Computation*, 240:308–325, 2014.

A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau. Sentiment Analysis of Twitter Data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38. Association for Computational Linguistics, 2011.

E. Agirre and A. Soroa. Personalizing Pagerank for Word Sense Disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 33–41. Association for Computational Linguistics, 2009.

S. Amir, M. B. Almeida, B. Martins, J. a. Filgueiras, and M. J. Silva. TUGAS: Exploiting unlabelled data for Twitter sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 673–677, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

S. Amir, R. Astudillo, W. Ling, M. J. Silva, and I. Trancoso. INESC-ID at SemEval-2016 Task 4-A: Reducing the Problem of Out-of-Embedding Words. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 238–242, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1036`.

R. Astudillo, S. Amir, W. Ling, B. Martins, M. J. Silva, and I. Trancoso. INESC-ID: Sentiment Analysis without Hand-Coded Features or Linguistic Resources using Embedding Subspaces. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 652–656, Denver, Colorado, June 2015. Association for Computational Linguistics.

S. Asur and B. A. Huberman. Predicting the future with social media. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 492–499. IEEE, 2010.

S. Baccianella, A. Esuli, and F. Sebastiani. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *Proceedings of the Seventh International Conference of Language Resources and Evaluation*, volume 10, pages 2200–2204, 2010.

A. Bakliwal, P. Arora, S. Madhappan, N. Kapre, M. Singh, and V. Varma. Mining Sentiments from Tweets. *Proceedings of the WASSA*, 12, 2012.

I. Bernardo, R. Henriques, and V. Lobo. Social Market: Stock Market and Twitter Correlation. In *International Conference on Intelligent Decision Technologies*, pages 341–356. Springer, 2017.

J. Bollen, H. Mao, and X. Zeng. Twitter Mood Predicts the Stock Market. *Journal of Computational Science*, 2(1):1–8, 2011.

B. Bonev, G. Ramírez-Sánchez, and S. O. Rojas. Opinum: Statistical Sentiment Analysis for Opinion Classification. In *Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis*, pages 29–37. Association for Computational Linguistics, 2012.

N. N. Bora. Summarizing Public Opinions in Tweets. *Journal Proceedings of CICLing*, 2012.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, et al. Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition. In *International conference on pattern recognition*, pages 77–77. IEEE Computer Society Press, 1994.

L. Breiman. Bagging Predictors. *Machine learning*, 24(2):123–140, 1996.

P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-Based n-gram Models of Natural Language. *Computational linguistics*, 18(4):467–479, 1992.

N. Chambers, V. Bowen, E. Genco, X. Tian, E. Young, G. Harihara, and E. Yang. Identifying Political Sentiment between Nation States with Social Media. In *EMNLP*, pages 65–75, 2015.

M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

S. F. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

C. Cortes and V. Vapnik. Support-Vector Networks. *Machine learning*, 20(3):273–297, 1995.

M. Costache, M. Liénou, and M. Datcu. On Bayesian Inference, Maximum Entropy and Support Vector Machines Methods. In *Aip Conference Proceedings*, volume 872, pages 43–51. AIP, 2006.

J. Deriu, M. Gonzenbach, F. Uzdilli, A. Lucchi, V. De Luca, and M. Jaggi. SwissCheese at SemEval-2016 Task 4: Sentiment Classification Using an Ensemble of Convolutional Neural Networks with Distant Supervision. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1124–1128, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1173`.

N. A. Diakopoulos and D. A. Shamma. Characterizing Debate Performance via Aggregated Twitter Sentiment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1195–1198. ACM, 2010.

R. Dijkman, P. Ipeirotis, F. Aertsen, and R. van Helden. Using Twitter to Predict Sales: A Case Study. *arXiv preprint arXiv:1503.04599*, 2015.

L. Dong, F. Wei, Y. Yin, M. Zhou, and K. Xu. Splusplus: A Feature-Rich Two-stage Classifier for Sentiment Analysis of Tweets. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 515–519, Denver, Colorado, June 2015. Association for Computational Linguistics.

J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

S. Ebert, N. T. Vu, and H. Schütze. A Linguistically Informed Convolutional Neural Network. In *6TH Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA 2015*, page 109, 2015.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

A. L. Firmino Alves, C. d. S. Baptista, A. A. Firmino, M. G. d. Oliveira, and A. C. d. Paiva. A Comparison of SVM Versus Naive-Bayes Techniques for Sentiment Analysis in Tweets: A Case Study with the 2013 FIFA Confederations Cup. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, pages 123–130. ACM, 2014.

J. L. Fleiss. Measuring Nominal Scale Agreement Among Many Raters. *Psychological bulletin*, 76(5):378, 1971.

J. Friedman. Another Approach to Polychotomous Classification. Technical report, Technical report, Department of Statistics, Stanford University, 1996.

J. Garten, K. Sagae, V. Ustun, and M. Dehghani. Combining Distributed Vector Representations for Words. In *Proceedings of NAACL-HLT*, pages 95–101, 2015.

K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

S. Giorgis, A. Rousas, J. Pavlopoulos, P. Malakasiotis, and I. Androutsopoulos. aueb.twitter.sentiment at SemEval-2016 Task 4: A Weighted Ensemble of SVMs for Twitter Sentiment Analysis. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 96–99, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1012`.

X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

A. Go, R. Bhayani, and L. Huang. Twitter Sentiment Classification Using Distant Supervision. *CS224N Project Report, Stanford*, pages 1–12, 2009.

R. González-Ibánez, S. Muresan, and N. Wacholder. Identifying Sarcasm In Twitter: A Closer Look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 581–586. Association for Computational Linguistics, 2011.

C. Grier, K. Thomas, V. Paxson, and M. Zhang. @ Spam: The Underground on 140 Characters or Less. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 27–37. ACM, 2010.

T. Günther and L. Furrer. Gu-mlt-lt: Sentiment Analysis of Short Messages Using Linguistic Features and Stochastic Gradient Descent. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in conjunction with the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, volume 2, pages 328–332, 2013.

T. Günther, J. Vancoppenolle, and R. Johansson. RTRGO: Enhancing the GU-MLT-LT System for Sentiment Analysis of Short Messages. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 497–502, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

M. Hagen, M. Potthast, M. Büchner, and B. Stein. Webis: An Ensemble for Twitter Sentiment Detection. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 582–589, Denver, Colorado, June 2015. Association for Computational Linguistics.

N. Haldenwang. Twitter Sentiment Analysis: On Feature Engineering, Classifier Performance and Realtime Tracking. `http://www2.inf.uos.de/prakt/pers/dipl/nhaldenwang_msc.pdf`, 2013.

N. Haldenwang and O. Vornberger. Sentiment Uncertainty and Spam in Twitter Streams and Its Implications for General Purpose Realtime Sentiment Analysis. In *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology (GSCL-2015)*, pages 157–159, 2015.

N. Haldenwang, K. Ihler, J. Kniephoff, and O. Vornberger. A Comparative Study of Uncertainty Based Active Learning Strategies for General Purpose Twitter Sentiment

Analysis with Deep Neural Networks. In *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology (GSCL-2017)*, page to appear, 2017.

H. Hamdan, P. Bellot, and F. Bechet. Lsislif: Feature Extraction and Label Weighting for Sentiment Analysis in Twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 568–573, Denver, Colorado, June 2015. Association for Computational Linguistics.

J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan kaufmann, 2006.

S. Han and R. Kavuluru. On Assessing the Sentiment of General Tweets. In *Canadian Conference on Artificial Intelligence*, pages 181–195. Springer, 2015.

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

S. Haykin. Neural Networks: A Comprehensive Foundation. 1994.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9 (8):1735–1780, 1997.

C.-W. Hsu and C.-J. Lin. A Comparison of Methods for Multiclass Support Vector Machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

M. Hu and B. Liu. Mining and Summarizing Customer Reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

D. Inouye and J. K. Kalita. Comparing Twitter Summarization Algorithms for Multiple Post Summaries. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 298–306. IEEE, 2011.

B. J. Jansen, M. Zhang, K. Sobel, and A. Chowdury. Twitter Power: Tweets as Electronic Word of Mouth. *Journal of the American society for information science and technology*, 60(11):2169–2188, 2009.

T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Technical report, DTIC Document, 1996.

T. Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Springer, 1998.

T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer Academic Publishers, 2002.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

R. M. Karampatsis, J. Pavlopoulos, and P. Malakasiotis. AUEB: Two Stage Sentiment Analysis of Social Network Messages. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 114–118, 2014.

Y. Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751, 2014.

S. Kiritchenko, X. Zhu, and S. M. Mohammad. Sentiment Analysis of Short Informal Texts. *Journal of Artificial Intelligence Research*, pages 723–762, 2014.

N. Kökciyan, A. Celebi, A. Ozgür, and S. Usküdarlı. BOUNCE: Sentiment Classification in Twitter using Rich Feature Sets. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in conjunction with the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, volume 2, pages 554–561, 2013.

T. Koo, X. Carreras Pérez, and M. Collins. Simple Semi-Supervised Dependency Parsing. In *46th Annual Meeting of the Association for Computational Linguistics*, pages 595–603, 2008.

U. H.-G. Kreßel. Pairwise Classification and Support Vector Machines. In *Advances in kernel methods*, pages 255–268. MIT Press, 1999.

J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *biometrics*, pages 159–174, 1977.

Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient Backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

K. Lee, B. D. Eoff, and J. Caverlee. Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter. In *In AAAI Int'l Conference on Weblogs and Social Media (ICWSM)*. Citeseer, 2011.

D. D. Lewis and W. A. Gale. A Sequential Algorithm for Training Text Classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

Z. Li, Z. Xiong, Y. Zhang, C. Liu, and K. Li. Fast Text Categorization Using Concise Semantic Analysis. *Pattern Recognition Letters*, 32(3):441–448, 2011.

C. Liebrecht, F. Kunneman, and A. van den Bosch. The Perfect Solution for Detecting Sarcasm in Tweets# Not. pages 29–37, 2013.

P.-C. Lin and P.-M. Huang. A Study of Effective Features for Detecting Long-Surviving Twitter Spam Accounts. In *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, pages 841–846. IEEE, 2013.

W. Ling, C. Dyer, A. Black, and I. Trancoso. Two/Too Simple Adaptations of word2vec for Syntax Problems. *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies*, 2015.

K.-L. Liu, W.-J. Li, and M. Guo. Emoticon Smoothed Language Models for Twitter Sentiment Analysis. In *AAAI*, 2012.

M. Mccord and M. Chuah. Spam Detection on Twitter Using Traditional Classifiers. In *International Conference on Autonomic and Trusted Computing*, pages 175–186. Springer, 2011.

R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, and S. Petrovic. Scalable Distributed Event Detection for Twitter. In *Big Data, 2013 IEEE International Conference on*, pages 543–549. IEEE, 2013.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

J. Mitchell and M. Lapata. Composition in Distributional Models of Semantics. *Cognitive science*, 34(8):1388–1429, 2010.

Y. Miura, S. Sakaki, K. Hattori, and T. Ohkuma. TeamX: A Sentiment Analyzer with Enhanced Lexicon Mapping and Weighting Scheme for Unbalanced Data. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 628–632, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

S. M. Mohammad and P. D. Turney. Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 26–34. Association for Computational Linguistics, 2010.

S. M. Mohammad, S. Kiritchenko, and X. Zhu. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in conjunction with the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, volume 2, pages 321–327, 2013.

V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

P. Nakov, S. Rosenthal, A. Ritter, and T. Wilson. SemEval-2013 Task 2 : Sentiment Analysis in Twitter. volume 2, pages 312–320, 2013.

P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov. SemEval-2016 Task 4: Sentiment Analysis in Twitter. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1–18, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1001`.

N. Neubauer. *Semantik und Sentiment: Konzepte, Verfahren und Anwendungen von Text-Mining*. Dissertation, Universität Osnabrück, 2014. URL `https://repositorium.uni-osnabrueck.de/handle/urn:nbn:de:gbv:700-2014060612524`.

N. Neubauer, N. Haldenwang, and O. Vornberger. The Bidirectional Co-occurrence Measure: A Look at Corpus-based Co-occurence Statistics and Current Test Sets. In *Proceedings of the 6th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, 2013.

G. Neumann and S. Schmeier. Combining Shallow Text Processing and Machine Learning in Real World Applications. In *Proceedings of the IJCAI-99 workshop on Machine Learning for Information Filtering, Stockholm, Sweden*, 1999.

F. Å. Nielsen. A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs. In *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*, pages 93–98, 2011.

O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith. Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters. In *Proceedings of NAACL-HLT*, pages 380–390, 2013.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999.

A. Pak and P. Paroubek. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In *LREC*, 2010.

G. Paltoglou and M. Thelwall. A Study of Information Retrieval Weighting Schemes for Sentiment Analysis. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1386–1395. Association for Computational Linguistics, 2010.

B. Pang, L. Lee, and S. Vaithyanathan. Thumbs Up?: Sentiment Classification Using Machine Learning Techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

S. Patra and L. Bruzzone. A Cluster-Assumption Based Batch Mode Active Learning Technique. *Pattern Recognition Letters*, 33(9):1042–1048, 2012.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, volume 14, pages 1532–1543, 2014.

S. Petrović, M. Osborne, and V. Lavrenko. Streaming First Story Detection With Application to Twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 181–189. Association for Computational Linguistics, 2010.

S. Poria, E. Cambria, and A. Gelbukh. Deep Convolutional Neural Network Textual Features and Multiple Kernel Learning for Utterance-Level Multimodal Sentiment Analysis. In *Proceedings of EMNLP*, pages 2539–2544, 2015.

M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

T. Proisl, P. Greiner, S. Evert, and B. Kabashi. KLUE: Simple and Robust Methods for Polarity Classification. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in conjunction with the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, volume 2, pages 395–401, 2013.

J. Read. Using Emoticons to Reduce Dependency in Machine Learning Techniques for Sentiment Classification. In *Proceedings of the ACL Student Research Workshop*, pages 43–48. Association for Computational Linguistics, 2005.

H. Reckman, C. Baird, J. Crawford, R. Crowell, L. Micciulla, S. Sethi, and F. Veress. teragram: Rule-Based Detection of Sentiment Phrases Using SAS Sentiment Analysis. 2:513–519, 2013.

R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

X. Rong. word2vec Parameter Learning Explained. *arXiv preprint arXiv:1411.2738*, 2014.

S. Rosenthal, A. Ritter, P. Nakov, and V. Stoyanov. SemEval-2014 Task 9: Sentiment Analysis in Twitter. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 73–80, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

S. Rosenthal, P. Nakov, S. Kiritchenko, S. Mohammad, A. Ritter, and V. Stoyanov. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 451–463, Denver, Colorado, June 2015. Association for Computational Linguistics.

D. Roth and D. Zelenko. Part of Speech Tagging Using a Network of Linear Separators. In *Coling-Acl, The 17th International Conference on Computational Linguistics*, pages 1136–1142, 1998. URL `http://cogcomp.cs.illinois.edu/papers/pos.pdf`.

M. Rouvier and B. Favre. SENSEI-LIF at SemEval-2016 Task 4: Polarity embedding fusion for robust sentiment analysis. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 202–208, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1030`.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. Technical report, DTIC Document, 1985.

J. Ruppenhofer and I. Rehbein. Semantic Frames as an Anchor Representation for Sentiment Analysis. In *Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis*, pages 104–109. Association for Computational Linguistics, 2012.

M. Sahlgren. The Distributional Hypothesis. *Italian Journal of Linguistics*, 20(1): 33–54, 2008.

H. Saif, Y. He, and H. Alani. Alleviating Data Sparsity for Twitter Sentiment Analysis. In *The 2nd Workshop on Making Sense of Microposts*, 2012a.

H. Saif, Y. He, and H. Alani. Semantic Sentiment Analysis of Twitter. In *The Semantic Web–ISWC 2012*, pages 508–524. Springer, 2012b.

R. E. Schapire. The Strength of Weak Learnability. *Machine learning*, 5(2):197–227, 1990.

B. Settles. *Curious Machines: Active Learning with Structured Instances*. PhD thesis, University of Wisconsin–Madison, 2008.

B. Settles. Active Learning Literature Survey. *University of Wisconsin, Madison*, 52 (55-66):11, 2010.

B. Settles and M. Craven. An Analysis of Active Learning Strategies for Sequence Labeling Tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics, 2008.

A. Severyn and A. Moschitti. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 464–469, Denver, Colorado, June 2015. Association for Computational Linguistics.

A. Simsek and P. Karagoz. Sentiment Enhanced Hybrid TF-IDF for Microblogs. In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 311–317. IEEE, 2014.

J. Smailović, J. Kranjc, M. Grčar, M. Žnidaršič, and I. Mozetič. Monitoring the Twitter Sentiment During the Bulgarian Elections. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

R. Socher, E. H. Huang, J. Pennin, C. D. Manning, and A. Y. Ng. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems*, pages 801–809, 2011.

A. Søgaard and A. Johannsen. Robust Learning in Random Subspaces: Equipping NLP for OOV Effects. In *24th International Conference on Computational Linguistics*, pages 1171–1180, 2012.

J. Song, S. Lee, and J. Kim. Spam Filtering in Twitter Using Sender-Receiver Relationship. In *International Workshop on Recent Advances in Intrusion Detection*, pages 301–317. Springer, 2011.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

P. J. Stone, D. C. Dunphy, and M. S. Smith. The General Inquirer: A Computer Approach to Content Analysis. 1966.

O. Täckström, R. McDonald, and J. Uszkoreit. Cross-Lingual Word Clusters for Direct Transfer of Linguistic Structure. In *Proceedings of the 2012 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 477–487. Association for Computational Linguistics, 2012.

M. Taddy. Document Classification by Inversion of Distributed Language Representations. *arXiv preprint arXiv:1504.07295*, 2015.

D. Tang, F. Wei, B. Qin, T. Liu, and M. Zhou. Coooolll: A Deep Learning System for Twitter Sentiment Classification. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 208–212, Dublin, Ireland, August 2014a. Association for Computational Linguistics and Dublin City University.

D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1555–1565, 2014b.

A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. Storm@ Twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

A. Tumasjan, T. O. Sprenger, P. Sandner, and I. M. Welpe. Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. *ICWSM*, 10:178–185, 2010.

J. Turian, L. Ratinov, and Y. Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

P. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. 2001.

P. D. Turney and M. L. Littman. Measuring Praise and Criticism: Inference of Semantic Orientation from Association. *ACM Transactions on Information Systems (TOIS)*, 21(4):315–346, 2003.

V. Vapnik. Estimation of Dependencies Based on Empirical Data, translated by S. Kotz, 1982.

V. Vapnik. *The Nature of Statistical Learning Theory*. springer, 2000.

V. N. Vapnik. Statistical Learning Theory. 1998.

X. Wang, C. Zhang, Y. Ji, L. Sun, L. Wu, and Z. Bao. A Depression Detection Model Based on Sentiment Analysis In Micro-Blog Social Network. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 201–213. Springer, 2013.

J. Wiebe, T. Wilson, and C. Cardie. Annotating Expressions of Opinions and Emotions in Language. *Language resources and evaluation*, 39(2-3):165–210, 2005.

T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

S. Xu, H. Liang, and T. Baldwin. UNIMELB at SemEval-2016 Tasks 4A and 4B: An Ensemble of Neural Networks and a Word2Vec Based Model for Sentiment Classification. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 183–189, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1027`.

B. Yu and Z.-b. Xu. A Comparative Study for Content-Based Dynamic Spam Classification Using Four Machine Learning Algorithms. *Knowledge-Based Systems*, 21(4): 355–362, 2008.

H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual Coordinate Descent Methods for Logistic Regression and Maximum Entropy Models. *Machine Learning*, 85(1):41–75, 2011.

Q. Yuan, G. Cong, and N. M. Thalmann. Enhancing Naive Bayes with Various Smoothing Methods for Short Text Classification. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 645–646. ACM, 2012.

C. Zhai and J. Lafferty. A study of Smoothing Methods for Language Models Applied to Information Retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2): 179–214, 2004.

R. Zhang and A. I. Rudnicky. A Large Scale Clustering Scheme for Kernel k-Means. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 289–292. IEEE, 2002.

Y. Zhang and B. Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1510.03820*, 2015.

X. Zhu, S. Kiritchenko, and S. Mohammad. NRC-Canada-2014: Recent Improvements in the Sentiment Analysis of Tweets. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 443–447, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.