Leopold-Franzens-University Innsbruck

Institute of Computer Science
Databases and Information Systems

Real-time Twitter Sentiment Classification based on Apache Storm

Master Thesis

Martin Illecker

supervised by
Dr. Eva Zangerle

Innsbruck, March 18, 2015

Abstract

The main goal of this master's thesis is to integrate techniques of sentiment classification within a real-time processing system. Therefore, it presents an approach called *SentiStorm*, which is based on *Apache Storm* and uses different machine learning techniques to identify the sentiment of a tweet. *SentiStorm* uses Part-of-Speech (POS) tags, Term Frequency–Inverse Document Frequency (TF-IDF) and multiple sentiment lexica to extract a feature vector out of a tweet. This extracted feature vector is processed by a Support Vector Machine (SVM), which predicts the sentiment based on a trained dataset.
Finally, this thesis will present the evaluation of *SentiStorm* based on the Semantic Evaluation (SemEval) dataset of 2013. The quality evaluation shows that *SentiStorm* is comparable with state-of-art sentiment classification systems. In addition to its high prediction quality, the performance results proof the possibility to run this sentiment classification also in real-time.

Contents

Chapter 1

Introduction

Linguistics and natural language processing are very active research areas because of their appliance in many different domains. Especially sentiment analysis or opinion mining has recently become one of the most important research fields in natural language processing. Beyond the scientific research, sentiment analysis has also become a foundation for business decisions, which are based on real-time information. For example, core business processes such as market-sensing, customer acquisition or customer relationship management (CRM) rely on real-time information. Therefore, opinion mining in real-time is a promising application in the field of natural language processing.
Opinion mining tries to analyze the sentiment, opinion, attitude or emotion of a written language. The massively increased data of social media, tweets, reviews, blogs and much more can be the source for a sentiment analysis. A sentiment or opinion always is subjective and therefore it is important to use a collection of sentiments from many people. This huge amount of opinionated data also increases the importance of sentiment analysis [Liu12, pp. 1–2].
The techniques of sentiment analysis are used in many different fields such as data mining, web mining or text mining. But the results of sentiment analysis are also widely used outside the computer science field. For example, social or management sciences rely on opinion mining techniques to support their decision making [Liu12, pp. 1–2].
Techniques such as classification, regression or ranking can cover nearly all kinds of problems in opinion mining. Among these, classification is probably the most studied technique in sentiment analysis. For example, it can be used to classify tweets or reviews in positive, negative or neutral. State-of-the-art sentiment classification or regression systems mainly use supervised techniques [PL08, p. 15] [Liu12, p. 24]. Sentiment classification can be executed on different levels. For example, it can be applied on document, sentence or phrase level [Liu12, p. 9].

This master's thesis uses sentiment classification based on the document level for an entire tweet text. A tweet text with a maximum of 140 characters is very short and therefore users often include abbreviations or slang expressions. Tweets also consist of specific terms such as retweet, hashtags or @ to mention other Twitter users. Furthermore, users are able to include emoticons, which are often a good indicator for the overall sentiment.
Table 1.1 illustrates two example tweets together with their corresponding sentiment. The first tweet text contains two emoticons <3 and :) and the hashtag *#love*, which indicate an overall positive sentiment of the tweet. The negative sentiment of the second tweet can also be obtained by the hashtag *#sad* and the emoticon :(.

Table 1.1: Example tweets with sentiment

Tweet Text	**Sentiment**
@Hugo I love u !! <3 :) #love	positive
#sad Not going to carnival tomorrow :(http://t.co/abcdefg	negative

Summing up, the language used in tweets is completely different to the traditional language. This master's thesis uses sentiment classification to obtain the sentiment of a tweet in real-time.

1.1 Motivation

The first and probably most promising idea for this master's thesis is based on the increasing importance of real-time processing systems. The transformation of big data batch processing systems to real-time processing systems has already begun. The focus shifts from batch to real-time processing because of the increasing need to make business decisions based on real-time information. For example, business decisions require real-time market information, customer or financial data. Furthermore, continuous streams such as Twitter or event logs have to be processed in real-time. Instead of waiting minutes for a job on a large-scale distributed batch processing system such as *Apache Hadoop*, the results should be available just in time. Therefore, scalable real-time systems such as *Apache Storm* support the execution of millions of tuples per second. Real-time systems have many different use cases such as trend analysis, social media mining, advertising analysis, event monitoring and much more [And13, p. 1].
The second idea for this thesis is based on social media mining and in particular on sentiment analysis. The idea of running sentiment analysis in real-time is the main goal of this master's thesis. Sentiment analysis

has become an important research field in the area of natural language processing because of the increasing amount of opinionated data. Furthermore, opinion mining often is the foundation for business decisions, politics analysis and much more.

The objective of this master's thesis is to combine a real-time processing system with sentiment classification techniques. Therefore, the large-scalable real-time processing system *Apache Storm* is used. The sentiment classification algorithm is based on a Support Vector Machine (SVM) and tries to classify an infinite tweet stream based on its sentiments in real-time.

1.2 Overview of the document

The introduction in Chapter 1 provides a short overview of sentiment analysis and the motivation for this thesis. Chapter 2 explains current state-of-the-art supervised, unsupervised and combined techniques of sentiment classification. *Apache Storm* including its architecture and components are illustrated in Chapter 3. Chapter 4 presents the *SentiStorm* approach and explains the architecture of *SentiStorm*, its components and the feature extraction in detail. The *SentiStorm* approach is evaluated in Chapter 5, which consists of a quality and performance evaluation. Finally, Chapter 6 presents the related work in the field of sentiment classification and real-time processing. At the end, a conclusion summarizes the master's thesis.

Chapter 2

Sentiment Classification

This chapter presents different sentiment classification techniques. Sentiment classification is probably one of the most studied topic in the research field of natural language processing [Liu12, p. 23]. The goal of sentiment classification is to predict a positive, negative or neutral sentiment for a given document. The following sections present unsupervised, supervised and combined techniques for sentiment classification.

2.1 Unsupervised Techniques

Unsupervised sentiment classification techniques are mostly based on sentiment lexica. Therefore, they are also called lexicon-based methods. Each word of a given document is examined for its polarity by looking it up in a sentiment dictionary. Finally, the overall sentiment of a document is positive, if it contains more positive words than negatives, otherwise it is negative [Liu12, p. 29].
A sentiment dictionary is also called sentiment lexicon, opinion lexicon or polarity lexicon. It contains individual words or phrases, which are assigned by a sentiment score. For example, words like *amazing* or *good* have a positive sentiment score whereas *bad* or *awful* have a negative score [Liu12, p. 79]. The sentiment score of a word or phrase is usually specified by a number. For example, the AFINN [Nie11b] sentiment lexicon consists of sentiment scores between -5 (negative) and 5 (positive). The scores within this range map to different strengths of positive and negative sentiments. For example, the word *awful* has a sentiment score of -4 and therefore is more negative than the word *bad* with a score of -2. More sentiment lexica are explained in detail in Section 4.6.3.
The generation of sentiment lexica is mainly based on the following three approaches [Liu12, pp. 79]:

1. **Manual Approach**
   A manual sentiment annotation is very time-consuming and labor-

intensive and therefore is usually used in combination with automated approaches such as the dictionary-based and corpus-based approach. It can be executed after an automated approach to verify the results and correct possible errors [Liu12, p. 79].

2. **Dictionary-based Approach**
   A dictionary-based approach automatically generates a sentiment lexicon based on a dictionary that contains synonyms and antonyms for each word such as *WordNet* [MBF$^+$90]. *WordNet* is a lexical database, which consists of English nouns, verbs, adjectives and adverbs. It is organized by synonyms [MBF$^+$90, p. 1]. There are many different dictionary-based approaches available, the two most important approaches are presented below [Liu12, pp. 80].
   The first dictionary-based approach consists of two steps. In the first step, the sentiment for a small set of words is manually annotated. In the next step, the approach searches the dictionary (e.g., *WordNet*) for synonyms and antonyms and adds the resulting words including their sentiment score to the initial set of words. This second step is repeated until no more new words can be found. Finally, a manual review should fix possible errors of this automatic approach [Liu12, p. 80].
   Another dictionary-based approach uses a distance-based method to obtain the sentiment of a word. The semantic orientation of a term can be defined by its relative distance to two predefined reference terms. The distance between two terms $d(t_1, t_2)$ can be calculated by the length of the shortest path, which connects t_1 and t_2. For example, the semantic orientation (SO) of term t based on the reference terms *good* and *bad* is defined by the following equation [Liu12, p. 80]:
   $$SO(t) = \frac{d(t, \text{bad}) - d(t, \text{good})}{d(\text{good}, \text{bad})}$$
   The semantic orientation is positive if and only if $SO(t) > 0$, otherwise it is negative. The absolute value of $SO(t)$ represents the strength of the positive or negative sentiment [Liu12, p. 80].

3. **Corpus-based Approach**
   A corpus-based approach is mainly used for the following two scenarios [Liu12, p. 83]:
   (a) Generate a new sentiment lexicon for a specific domain corpus based on a given list of sentiment words
   (b) Generate a new sentiment lexicon for a specific domain by adapting a general-purpose sentiment lexicon

   Corpus-based approaches are very difficult, because the sentiment of a word can be positive in one context but negative in another.

> One key idea of this approach is to use a set of linguistic rules to obtain new sentiment words from a specific corpus. For example, the conjunction rule defines that conjoined adjectives usually have the same orientation. If *beautiful* is known as positive in a given sentiment word list, then *young* would also become a positive sentiment based on the AND conjunction in "young *and* beautiful" [Liu12, p. 83].

A different unsupervised technique was presented by Turney [Tur02] in 2002. He used fixed syntactic patterns, which were based on part-of-speech (POS) tags, to predict the semantic orientation of a document. POS tags represent the grammatical class of a word such as noun, verb, adjective or adverb. For example, the POS tags according to the most commonly used Penn Treebank [San90] tagset for the given text "I love you" are "I/*PRP* love/*VBP* you/*PRP*". In this example, *PRP* stands for a personal pronoun and *VBP* for a present tense verb [San90, p. 4]. The complete Penn Treebank tagset is illustrated in Table 1 in the appendix and POS tagging is presented in detail in Section 4.5. Turney proposed fixed syntactic patterns for example, the first word has to be an adjective (*JJ*), the second word a singular or plural noun (*NN* or *NNS*) and the third word can be anything. If a phrase matches this syntactic pattern, then the semantic orientation is calculated by the point-wise mutual information of the matching terms. The average semantic orientation of all matching phrases determines the overall sentiment of a document. The goal of this approach was to recommend or not recommend a review based on the occurrences and order of adjectives and verbs.
Table 2.1 illustrates the example tweets of Table 1.1 and its lexicon-based sentiment classification. Words that are known by the sentiment lexicon are marked by [...] and followed by its positive (+) or negative (−) sentiment.

Table 2.1: Lexicon-based sentiment classification on example tweets

Tweet Text	**Sentiment**
@Hugo I **[love]**+ u !! **[<3]**+ **[:)]**+ #**[love]**+	positive
#**[sad]**− Not going to carnival tomorrow **[:(]**− http://t.co/abcdefg	negative

The overall sentiment of the tweets in Table 2.1 is obtained by looking up each word in a sentiment lexicon. The first tweet text contains the word *love* and emoticons <3 and :), which are associated with a positive sentiment. Therefore, the overall sentiment is determined as positive. The sentiment of the second tweet is set negative because of the occur-

rences of *sad* and :(, which are known to have a negative sentiment.

In contrast to supervised techniques, unsupervised techniques do not require any annotated training data and training of a classifier.

2.2 Supervised Techniques

Supervised sentiment classification is a text classification problem and therefore, classifiers such as Naive Bayes, Maximum Entropy (MaxEnt) and Support Vector Machines (SVM) can be applied. Usually a two-class classification with positive and negative or a three-class classification including neutral is used [Liu12, p. 24]. The SVM classifier is explained in detail in Section 4.7. A classifier requires annotated training data, which consists of labelled documents. The labelling of these documents can be done manually or be automatically received from user-generated opinionated data such as reviews. After the classifier has been trained with the labelled data, it is able to predict the label for a new document based on the trained data. Therefore, a major difference between unsupervised and supervised techniques is the training phase and labelled training data, which are required by a classifier. Classifiers operate with feature vectors, which have to be generated for each document. An n-dimensional feature vector consists of n numerical features. A set of effective features is critical for the accuracy of a classifier [Liu12, pp. 24–25].
A feature set typically contains multiple different features. The most important features are explained below [Liu12, p. 25]:

1. **Terms and their frequency**
   Terms can be individual words (unigrams) or n-grams. An n-gram is a sequence of n terms. Bigrams ($n = 2$) and trigrams ($n = 3$) are often used in sentiment classification to recognize the negation of terms. Each term is associated with its frequency counts and sometimes also with its position. The bag-of-word model consists of n-grams and their frequency, but the position does not matter. An extension to the bag-of-word model is the weighting scheme Term Frequency–Inverse Document Frequency (TF–IDF). TF–IDF weights the term frequency by the importance of a term in all documents. This is explained in detail in Section 4.6.1. Terms and their frequency are essential features in an effective feature set [Liu12, p. 25].

2. **Part-of-Speech (POS)**
   POS tags of terms can be used as features as well. For example, adjectives can be handled separately, because they often indicate

a sentiment [Liu12, p. 25]. POS tags cannot only be included in bag-of-words, but they also can express of how many nouns, verbs, adjectives or adverbs a document consists.

3. **Sentiment words and phrases**
   Sentiment features indicate the occurrence of positive or negative terms in a document. The sentiment of words or phrases can be obtained from a sentiment lexicon. Emoticons are also sentiment words, because they often indicate the sentiment of the document. Beside the terms and their frequency, sentiment features are essential for sentiment classification.

Table 2.2 illustrates the example tweets of Table 1.1 and their corresponding feature vectors.

Table 2.2: Feature vectors of example tweets

Tweet Text	**Feature Vectors**	
@Hugo I **love** u !! **<3 :)** **#love**	$[2,1,1,0,0,0,0,0,0]$ $[1,1,0,0]$ $[4,0]$	Bag-of-words POS tags Sentiment
#sad Not going to carnival tomorrow **:(** http://t.co/abcdefg	$[0,0,0,1,1,1,1,1,1]$ $[2,1,1,1]$ $[0,2]$	Bag-of-words POS tags Sentiment

The features of Table 2.2 are composed of bag-of-words, POS tags and sentiment features. After the elimination of stop words $\{\mathrm{I}, \mathrm{u}, \mathrm{to}\}$ the bag-of-words for the example tweets are defined by the following sets:

$$\{(\mathrm{love}, 2), (<3, 1), (:), 1)\}$$
$$\{(\mathrm{sad}, 1), (\mathrm{not}, 1), (\mathrm{going}, 1), (\mathrm{carnival}, 1), (\mathrm{tomorrow}, 1), (:(, 1)\}$$

Users and URLs are ignored but the hashtags are extracted. The bag-of-words feature vector consists of the frequency count of every term in the union of these two sets $[\mathrm{tf}(\mathrm{love}), \mathrm{tf}(<3), ..., \mathrm{tf}(:()]$. For example, the term frequency of *love* is assigned to the first index of the bag-of-words feature vector. The POS tags feature vector consists of the frequency of POS tags. In this example, the feature vector includes the number of nouns, verbs, adjectives and adverbs. The first example tweet has one noun *love* and one verb *love*, but no adjective or adverb. The feature vector of the second tweet is $[2, 1, 1, 1]$, which represents two nouns *carnival* and *tomorrow*, one adjective *sad* and one adverb *not*. Finally, the sentiment features represent the number of positive and negative sentiments. The first tweet has four positive but no negative sentiments, which is represented by the feature vector $[4, 0]$. Based on

the combination of these feature vectors, a trained classifier is able to predict the sentiment of tweets.

2.3 Combined Techniques

Combined sentiment classification techniques combine supervised and unsupervised methods. For example, Liu et al. [LLLY04] proposed a combined technique, which consists of multiple steps. In the first step the given unlabeled documents are clustered. Then a feature selection is performed on these clusters. After a set of representative words for each cluster has been selected, the initial set of documents is labelled and used by a Naive Bayesian classifier.
Another interesting combined technique was published by Martín-Valdivia et al. [MVMCPOUnL13]. They introduced a meta-classifier, which combines supervised and unsupervised learning. The goal of this approach is to support the parallel use of multiple corpora such as Spanish and English. For each corpus there exists a supervised method. But non-English corpora are translated to English to support also an unsupervised lexicon-based method. The final meta-classifier is able to combine several features from the supervised and unsupervised models.

Chapter 3

Apache Storm

Apache Storm is a free Java and Clojure based framework that supports distributed real-time computations. Clojure is a functional programming language, which is based on Lisp and can be executed on the Java Virtual Machine (JVM). Storm was first developed by Nathan Marz at BackType, which was later acquired by Twitter in 2011. In 2014, Storm graduated to an Apache top-level project and became an open source framework. Storm supports the development of applications, which are processing large amount of data in real-time. In contrast to the batch-processing framework Hadoop, Storm has become an important platform for real-time processing [Gro15a] [JN14, p. 7].
Storm includes multiple features such as horizontal scalability, fault tolerance, guaranteed data processing and the support of different programming languages. The scalability feature of Storm includes the possibility to rebalance a cluster when new working nodes have been added. Guaranteed data processing ensures that if a worker node fails, Storm will automatically reassign its tasks and also replay all tuples to guarantee its processing. In addition to JVM-based languages, Storm supports also all programming languages that are able to read and write to the standard input and output. But one of the most important features of Storm is the real-time execution capability. A performance benchmark achieved over one million tuples per second per node [JN14, pp. 8–9] [LES12, pp. 3–4].
The following section introduces the architecture of Storm. The different levels of parallelism in Storm are explained in Section 3.2. Finally, the Storm topology and its components Spouts and Bolts are presented at the end of this chapter.

3.1 Architecture

The architecture of Storm is based on a master and slave pattern similar to the *Apache Hadoop* architecture. The master and slave processes are coordinated by a third component so-called *ZooKeeper*.

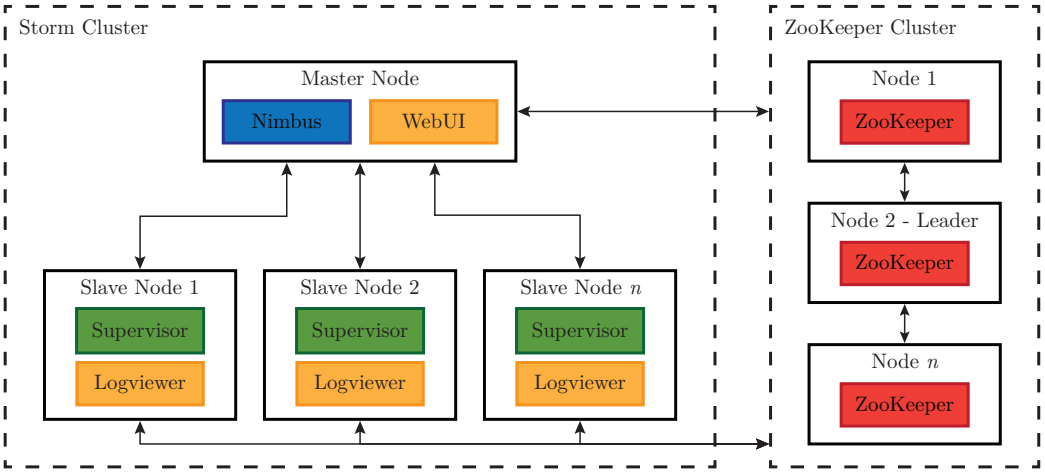

Figure 3.1: *Apache Storm* Architecture

Figure 3.1 illustrates the interaction between the Storm cluster and the *ZooKeeper* cluster. The architecture of Storm allows only a single master node. The master node consists of the *Nimbus* daemon and an optional WebUI daemon. The WebUI component provides a web-based user interface for the currently running tasks and cluster information. Each slave node includes exactly one *Supervisor* daemon and an optional *Logviewer* process. The *Logviewer* provides a web-based user interface for the logs of the slave node. The *ZooKeeper* cluster consists of $(2n + 1)$ *ZooKeeper* nodes, where one node is identified as the leader. At the startup of a topology the *Nimbus* daemon distributes the jar files directly to its worker nodes.
The architecture of Storm consists of the following three major components [JN14, pp. 10–11]:

1. ***Nimbus* (Master)**
   The *Nimbus* daemon represents the master component in the architecture of Storm and distributes the work among multiple workers. It assigns a task to a slave node, monitors the task progress and reschedules the task to another worker node, if a slave node fails. In contrast to the *JobTracker* in Hadoop, the *Nimbus* component is stateless because it stores all its data in the *ZooKeeper* cluster, which is the third major component of Storm. This avoids the

problem of a single point of failure. If a *Nimbus* daemon fails, it can be restarted without influencing any running task. The architecture of Storm allows only a single *Nimbus* component.

2. ***Supervisor* (Slave)**
   The architecture of Storm may contain one or more *Supervisor* nodes. A *Supervisor* is the slave or worker node and executes the actual tasks. It starts a new Java Virtual Machine (JVM) instance for each worker process. Similar to the *Nimbus* component, the *Supervisor* stores its data in the *ZooKeeper* cluster and therefore is a fail-fast process. A *Supervisor* daemon in the Storm architecture normally runs multiple worker instances and each worker is able to spawn new threads. The levels of parallelism in Storm are further presented in the following Section 3.2.

3. ***ZooKeeper* (Coordination)**
   The *ZooKeeper* cluster optimally consists of three, five or $(2n+1)$ *ZooKeeper* nodes, which are coordinating the shared information between the *Nimbus* and *Supervisor* components. All states of the running tasks and the *Nimbus* and *Supervisor* daemon are stored in the *ZooKeeper* cluster. Therefore, the state-less components *Nimbus* and *Supervisor* are fail-fast and can be killed and restarted without interrupting the running topology.

3.2 Levels of Parallelism

Storm provides four different levels of parallelism in its architecture. The following levels are used to run a topology in Storm [Gro15b]:

1. *Supervisor* (Slave)
2. *Worker* (JVM)
3. *Executor* (Thread)
4. *Task*

First of all Storm supports the distribution of work among multiple slave nodes. Each one of these worker nodes runs a single *Supervisor* daemon. A *Supervisor* executes one or multiple *Worker* processes within a dedicated JVM instance. At the next level each *Worker* is able to use multiple *Executor* threads within its JVM process. And finally each *Executor* thread executes one or more *Tasks* serially. These different levels of parallelism in Storm are illustrated in Figure 3.2. This figure shows one *Supervisor* daemon, which includes n *Worker* processes. The maximum number of *Workers* is specified within the *Supervisor* configuration. Each *Worker* in Figure 3.2 consists of five *Executor* threads. By default Storm runs one *Task* per each *Executor* thread but an *Executer* might also execute multiple *Tasks* serially. The feature of having

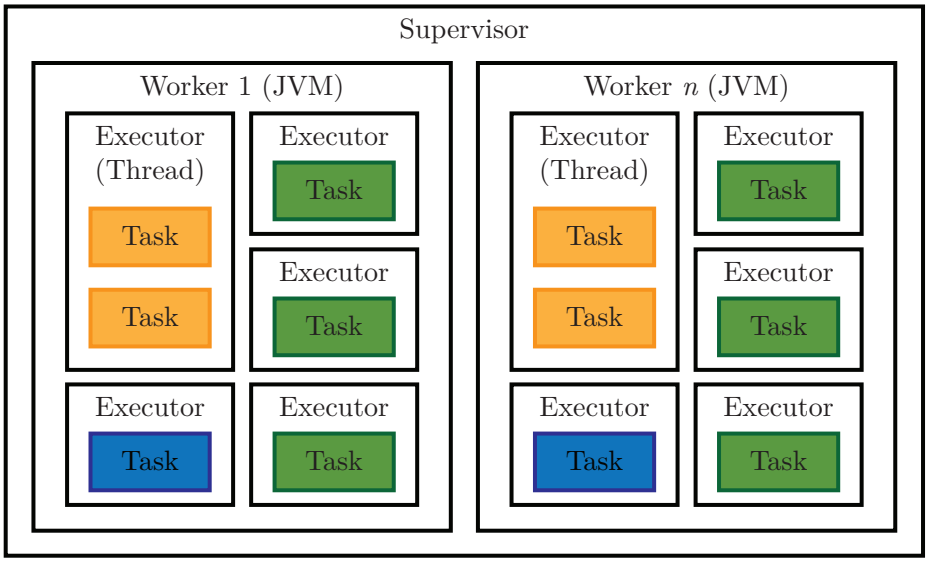

Figure 3.2: Levels of Parallelism in Storm

multiple *Tasks* that are executed in a serial fashion within an *Executor*, allows to test the parallelism of the system.

3.3 Spout

The terminology *Spout* defines the source of tuples in Storm. A *Spout* reads data from an external source and provides it to the Storm topology. For example, a *Spout* might listen to a Twitter stream and emits this data into a Storm stream. Storm supports reliable and unreliable *Spouts*. If a tuple fails during the execution within a topology, then a reliable *Spout* would replay it. In contrast, an unreliable *Spout* forgets a tuple as soon as it is emitted [Gro15c].
Listing 3.1 illustrates an example *Spout* implementation in Java. The most important method is `nextTuple`, which emits the next tuples into the Storm stream. The `open` method is called at the startup of the Spout and should contain all initialization. This example Spout infinitely emits zero values.

Listing 3.1: Example *Spout*

```
1 public class ExampleSpout extends BaseRichSpout {
2   private SpoutOutputCollector _collector;
3   @Override
4   public void open(Map conf, TopologyContext context,
        SpoutOutputCollector collector) {
5     _collector = collector;
6   }
```

```
7   @Override
8   public void nextTuple() {
9     _collector.emit(new Values(0));
10  }
11 }
```

3.4 Bolt

In Storm the actual processing of a task is executed by *Bolts*. A *Bolt* takes the tuples of one or multiple input streams, processes and emits them to one or multiple output streams. A *Bolt* can transform, filter, aggregate, join or execute other functions on tuples [Gro15c].
Listing 3.2 presents an example *Bolt* implementation in Java. The actual processing of tuples is done within the `execute` method. It is called for every tuple that comes from the *Bolt's* input stream. In this example the input value is incremented by one before it is emitted to the output stream. Since a tuple may contain multiple values, the identification of output fields can be specified within the `declareOutputFields` method. The `prepare` method is equivalent to the `open` method of a *Spout*. It is called at the startup of the *Bolt* and should include all initializations.

Listing 3.2: Example *Bolt*

```
1  public class ExampleBolt extends BaseRichBolt {
2    private OutputCollectorBase _collector;
3    @Override
4    public void declareOutputFields(OutputFieldsDeclarer
         declarer) {
5      declarer.declare(new Fields("incremented_tuple"));
6    }
7    @Override
8    public void prepare(Map conf, TopologyContext
       context, OutputCollectorBase collector) {
9      _collector = collector;
10   }
11   @Override
12   public void execute(Tuple input) {
13     int val = input.getInteger(0);
14     _collector.emit(input, new Values(val+1));
15     _collector.ack(input);
16   }
17 }
```

3.5 Topology

Storm uses its own terminology to describe the workflow. A topology in Storm defines a direct acyclic graph (DAG), which represents the structure and logic of a Storm real-time application. Each node in this DAG processes and forwards tuples in parallel [JN14, p. 11]. A topology typically consists of the two major components *Spouts* and *Bolts*.

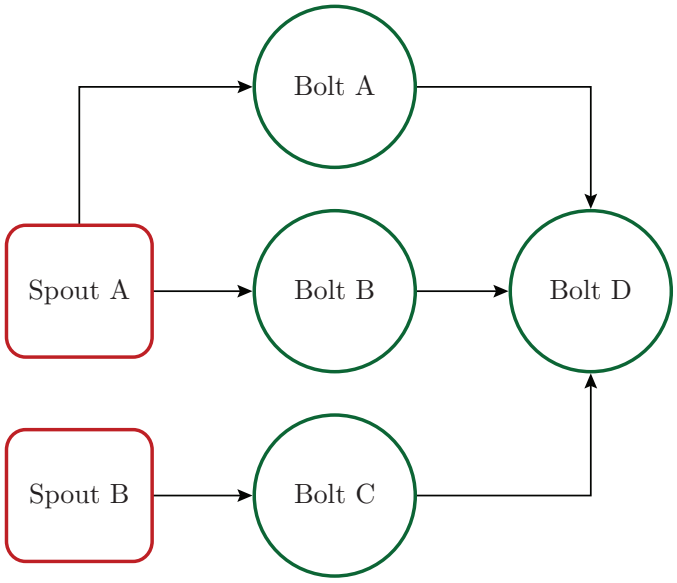

Figure 3.3: Graphical representation of a topology [JN14, p. 11]

Figure 3.3 illustrates a possible topology including two *Spouts* and four *Bolts*. The connection between *Spouts* and *Bolts* is called stream. A stream represents an infinite sequence of tuples, which can be processed in parallel. Storm supports different stream groupings, which specify how streams should be partitioned among different *Bolts*. [JN14, p. 11] [Gro15c].

Chapter 4

SentiStorm Approach

SentiStorm is a real-time sentiment classification system, which has been developed within the scope of this master's thesis. It is based on *Apache Storm* to provide real-time capabilities. *SentiStorm* uses natural language processing (NLP) techniques such as POS tagging and supervised sentiment classification to determine the sentiment of a tweet. It consists of Twitter-specific components such as *Tokenizer* and *Preprocessor* to address the different language used in tweets. The feature extraction component of *SentiStorm* uses TF-IDF, POS tags and multiple sentiment lexica to generate a feature vector of a tweet text. Finally, a trained Support Vector Machine (SVM) is able to predict the sentiment of the generated feature vector based on its training data. The concatenation of these NLP components is reflected in a Storm topology.
The following section presents the architecture of *SentiStorm* and its components. Then each component is explained in detail.

4.1 Architecture

The architecture of *SentiStorm* is aligned to a Storm topology. Every component is wrapped into a *Bolt*. The *SentiStorm* topology consists of one data source *Spout* and the following five *Bolts*:

1. *Tokenizer*
2. *Preprocessor*
3. *POS Tagger*
4. *Feature Vector Generation*
5. *Support Vector Machine (SVM)*

Figure 4.1 illustrates the topology of *SentiStorm* including its components. The *Dataset Spout* emits tweets from a local dataset into the Storm pipeline. It can be easily replaced by another *Spout*. For example, a *Twitter Spout* can be used to emit tweets directly from the real-

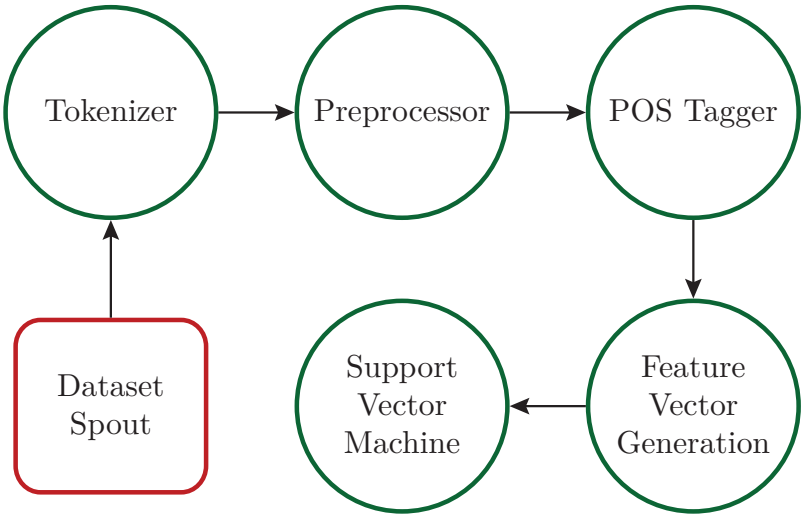

Figure 4.1: Topology of *SentiStorm*

time Twitter stream. After tweets have been emitted by a *Spout*, the *Tokenizer* replaces possible Unicode or HTML symbols and tokenizes the tweet text by a complex regular expression. Then, each token is processed by the *Preprocessor*, which tries to unify emoticons, fix slang language or gerund forms and remove elongations. The unification of emotions removes repeating characters to get a consistent set of emoticons. For example, the emoticon :-))) is replaced by :-) and therefore the sentiment can be easily obtained from an emoticon lexicon. Slang expressions such as *omg* are substituted by *oh my god* by the usage of multiple slang lexica. Gerund forms are fixed by checking the ending of words for an omitted *g* such as in *goin*. The remove elongations process is equivalent to the unification of emoticons and tries to eliminate repeating characters such as in *suuuper*. After the *Preprocessor*, a *POS Tagger* predicts the part-of-speech label for each token and forwards them to the *Feature Vector Generation*. The feature extraction process is a key component in *SentiStorm*. It generates a feature vector for each tweet based on the previously gathered data. The *Feature Vector Generation* component uses TF–IDF, POS tags and multiple sentiment lexica to map a tweet text into numerical features. Based on this feature vector the SVM component is finally able to predict the sentiment of the given tweet.
In the following sections each single component of *SentiStorm* is explained in detail. First, the data source of *SentiStorm*, which is implemented by a *Dataset Spout*, is described. Then each of the five *Bolts* of *SentiStorm*, which process the tweet and extract its sentiment, are discussed in detail.

4.2 Dataset Spout

The *Dataset Spout* is the source of the *SentiStorm* topology. At startup, it loads tweets into the main memory and emits them into the Storm stream.
Listing 4.1 illustrates the reliable *Dataset Spout* component. At the initialization, a set of tweets is loaded into the memory (line 13). Every time `nextTuple` is called, it will emit a tweet text and its identification (line 22). If all tweets have been emitted, then it will start with the first tweet again (line 19 to 21). This *Spout* is reliable because it assigns a global message identification to the tweet (line 22), which is later used by *Bolts* to acknowledge (ack) the tuple. If a *Bolt* fails or a tuple experiences a timeout, the guaranteed message processing of Storm will replay the tuple.

Listing 4.1: *Dataset Spout* (abbreviated)

```
1  public class DatasetSpout extends BaseRichSpout {
2    private SpoutOutputCollector m_collector;
3    private List<Tweet> m_tweets;
4    private int m_index = 0;
5    @Override
6    public void declareOutputFields(OutputFieldsDeclarer
         declarer) {
7      declarer.declare(new Fields("id", "text"));
8    }
9    @Override
10   public void open(Map config, TopologyContext context
       , SpoutOutputCollector collector) {
11       SpoutOutputCollector collector) {
12     m_collector = collector;
13     m_tweets = Dataset.getTweets();
14   }
15   @Override
16   public void nextTuple() {
17     Tweet tweet = m_tweets.get(m_index);
18     m_index++;
19     if (m_index >= m_tweets.size()) {
20       m_index = 0;
21     }
22     m_collector.emit(new Values(tweet.getId(), tweet.
         getText()), tweet.getId());
23   }
24 }
```

4.3 Tokenizer

The *Tokenizer* is the first *Bolt* in the *SentiStorm* topology and splits a tweet text into several tokens. In this process, the *Tokenizer* uses pattern matching with regular expressions. Furthermore, it replaces Unicode or HTML symbols before tokenizing the tweet text.

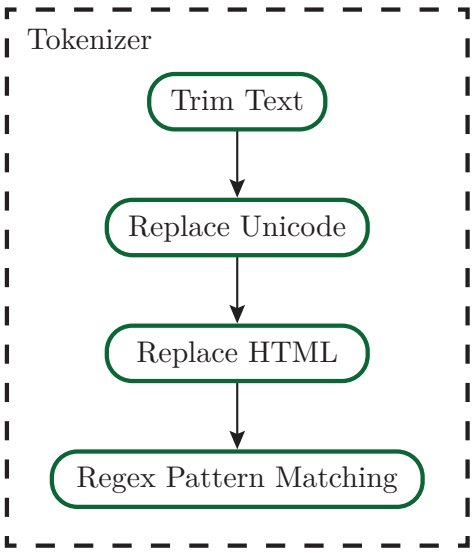

Figure 4.2: Workflow of the *Tokenizer*

Figure 4.2 illustrates the workflow of the *Tokenizer*. It simply starts by trimming the tweet text to remove leading and trailing white-spaces. In the next step, Unicode symbols are replaced. Table 4.1 presents an example set of Unicode symbols, which are used by *SentiStorm*. Unicode symbols are often used for emoticons within tweets. It is important to replace them to determine the sentiment of a tweet. After all Unicode symbols have been replaced, the *Tokenizer* also substitutes HTML symbols, which are primarily used for quotation marks. Some example HTML symbols are shown in Table 4.2.

Table 4.1: Examples of Unicode Symbols

Unicode	**Replacement**	**Description**
\u2019	’	Single quotation
\u002c	,	Comma
\uD83D\uDE03	:)	Smile
\uD83D\uDE12	:(	Unamused

Table 4.2: Examples of HTML Symbols

HTML	Replacement	Description
&#039;	’	Single quotation
&quot;	”	Double quotation
&#45;	-	Minus sign
&amp;	&	Ampersand

After every symbol has been replaced, pattern matching with regular expressions is applied to tokenize the tweet text. This regular expression consists of multiple patterns.
Listing 4.2 presents the regular expression pattern, which is used by the *Tokenizer*. First it tries to match an emoticon, followed by an URL or phone number. Then the *Tokenizer* tests to match an email address, username, hashtag and a lot more patterns. The regular expression finishes with `NOT_A_WHITESPACE`, which matches everything except a whitespace. The order of these individual patterns is very important.

Listing 4.2: *Tokenizer* Regular Expression

```
public static final Pattern TOKENIZER_PATTERN =
 Pattern.compile(EMOTICON + "|" + URL + "|" +
 PHONE + "|" + EMAIL_ADDRESS + "|" + USER_NAME + "|" +
 HASH_TAG + "|" + SLANG_PATTERN + "|" +
 ALTERNATING_LETTER_DOT + "|" +
 WORDS_WITH_APOSTROPHES_DASHES + "|" +
 SEPARATED_NUMBER + "|" + SPECIAL_NUMBER + "|" +
 WORDS_WITHOUT_APOSTROPHES_DASHES + "|" +
 ELLIPSIS_DOTS + "|" + NOT_A_WHITESPACE);
```

Listing 4.3 illustrates the `HASH_TAG` regular expression in detail. A hashtag has to start with `#` followed by one or more alphabetic characters. Then also numbers and special characters such as `_` or `-` are allowed. Finally, a hashtag must end with an alphabetic character, number or an underscore.

Listing 4.3: `HASH_TAG` Regular Expression

```
public static final String HASH_TAG = "(?:" +
  "\\#+([A-Za-z]+[A-Za-z0-9_\\’\\-]*[A-Za-z0-9_]+)" +
  ")";
```

Table 4.3 presents an example tweet and the corresponding output of the *Tokenizer*. The tweet text is tokenized based on the order of the regular expressions in Listing 4.2. Unicode and HTML symbols are substituted as well.

Table 4.3: *Tokenizer* output for the example tweet

Tweet Text	***Tokenizer* Output**
@user1 @user2 OMG I just watched Michael's comeback ! U remember him from the 90s ?? yaaaa \uD83D\uDE09 #michaelcomeback	[@user1, @user2, OMG, I, just, watched, Michael's, comeback, !, U, remember, him, from, the, 90, s, ?, ?, yaaaa, ;), #michaelcomeback]

4.4 Preprocessor

The *Preprocessor* component receives the tokenized tweet from the *Tokenizer* and prepares the tokens for the *POS Tagger*. Figure 4.3 illustrates the workflow of the *Preprocessor*, which consists of multiple steps.

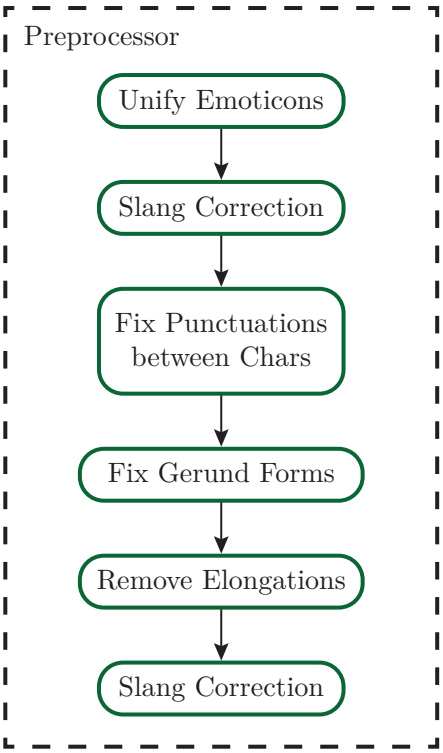

Figure 4.3: Workflow of the *Preprocessor*

In the first step, the *Preprocessor* unifies all emoticons. For example, the emoticon :-))) will become :-) to get a consistent set of emoticons. *SentiStorm* does currently not differentiate between these two emoticons, both of them have the same positive sentiment score based on the Sen-

tiStrength [The15] emoticons lexicon. Future extensions of *SentiStorm* might differentiate between these emoticons by using boost sentiment scores. In the second step, the *Preprocessor* tries to substitute slang expressions. The replacement of slang expressions will help the *POS Tagger* to determine the right POS tag. Table 4.4 presents an example list of slang expressions and their replacements.

Table 4.4: Examples of Slang Expressions

Slang	**Replacement**
w/	with
u	you
lol	laugh out loud
omg	oh my god

The next step fixes possible punctuations between characters. For example, the term *L.O.V.E* is replaced by the term *LOVE*. Punctuations between characters are eliminated by a regular expression, which is illustrated in Listing 4.4. The pattern matches a single alphabetic character, which is followed by one or more patterns of a dot and an additional single alphabetic character.

Listing 4.4: `ALTERNATING_LETTER_DOT` Regular Expression

```
1 public static final String ALTERNATING_LETTER_DOT = "[
    a-zA-Z]\\.(?:[a-zA-Z](\\.)?)+";
```

The *Preprocessor* also fixes incomplete gerund forms such as *goin* by replacing it with *going*. For that purpose, it uses the *WordNet* [MBF+90] dictionary to find a valid word. In the last step, elongations such as *suuuper* are removed. If an elongation has been removed by the *Preprocessor*, then it has to check the term for any slang expression again. The elimination of repeating characters is also based on a regular expression. Listing 4.5 illustrates a regular expression pattern, which matches three or more repeating characters. The pattern recognizes any character which is followed by itself at least two times and assigns it to a regular expression group.

Listing 4.5: `THREE_OR_MORE_REPEATING_CHARS` Regular Expression

```
1 public static final String
    THREE_OR_MORE_REPEATING_CHARS = "(.)\\1{2,}";
```

Table 4.5 illustrates the output of the *Preprocessor* based on the example tweet of Table 4.3. The slang expression *OMG* has been replaced by *Oh my God* and *U* has been replaced by *you*. The trickier term *yaaaa* was substituted to *ya* based on the remove elongations step and is finally recognized as slang expression and replaced by *yeah*.

Table 4.5: *Preprocessor* output for the example tweet

Tokenized Tweet	***Preprocessor* Output**
[@user1, @user2, OMG, I, just, watched, Michael's, comeback, !, U, remember, him, from, the, 90, s, ?, ?, yaaaa, ;), #michaelcomeback]	[@user1, @user2, Oh, my, God, I, just, watched, Michael's, comeback, !, you, remember, him, from, the, 90, s, ?, ?, yeah, ;), #michaelcomeback]

4.5 POS Tagger

The *POS Tagger* component determines the part-of-speech (POS) labels for the preprocessed tokens. Currently there are two major POS taggers available, which are highly specialized for the Twitter-specific language. The first POS tagger was presented by Derczynski et al. [DRCB13] of the General Architecture for Text Engineering (GATE) group at the University of Sheffield. Owoputi et al. [OOD+13] of the ARK research group at the Carnegie Mellon University proposed the second major POS tagger. Both taggers achieve a high accuracy of nearly 90%.
The GATE Twitter POS tagger is based on the commonly used Penn Treebank (PTB) tagset, which is illustrated in Table 1 in the appendix. In contrast, Owoputi et al. [OOD+13] used their own tagset, which is based on Gimpel et al. [GSO+11]. It is less descriptive than the PTB tagset. For example, the PTB tagset consists of many different adjective tags compared to only one adjective tag of the ARK tagset. The GATE Twitter POS tagger uses the Maximum Entropy (MaxEnt) Stanford tagger with their custom Twitter-specific model, compared to the ARK tagger, which is based on their own Maximum Entropy Markov Model (MEMM).
The first implementation of *SentiStorm* used the GATE POS tagger because of the commonly used PTB tagset support. But the major drawback in speed of the GATE tagger made a transition to the ARK tagger necessary. The GATE tagger is significantly slower than the ARK tagger and therefore it is not applicable in a real-time environment such as Storm. A performance comparison between the GATE and ARK tagger can be found in Section 5.2. The speed of the ARK tagger was highly optimized for social media analysis applications and achieves 10,000 tokens per second on a single CPU [OOD+13, p. 2].
Table 4.6 illustrates the ARK POS tagset, which is used by the *POS Tagger* component of *SentiStorm*. The tagset is grouped into nominal, Twitter specific, open-class words and closed-class words. The most important tags for *SentiStorm* are included in the group nominal, open-class words and Twitter specific.

Table 4.6: ARK POS tagset [GSO+11, p. 43]

Nominal

Tag	Description
N	common noun
O	pronoun
^	proper noun
S	nominal + possessive
Z	proper noun + possessive

Open-class words

Tag	Description
V	verb
A	adjective
R	adverb
!	interjection

Twitter specific

Tag	Description
#	hashtag
@	at-mention
~	discourse marker
U	URL or email address
E	emoticon

Closed-class words

Tag	Description
D	determiner
P	pre- or postposition
&	conjunction
T	verb particle
X	existential there, predeterminers

Miscellaneous

Tag	Description
$	numeral
,	punctuation
G	other abbreviations

Table 4.7 shows the preprocessed example tweet of Table 4.5 and the corresponding output of the *POS Tagger*, where the POS tags are attached to the terms. For example, the at-mentions of *user1* and *user2* are tagged by (@) and the term *Oh* is labelled as an interjection (!). The POS tagger also recognizes Twitter-specific expressions such as emoticons (E) and hashtags (#). These POS tags are used by the *Feature Vector Generation* component, which is explained in the following section.

Table 4.7: *POS Tagger* output for the example tweet

Preprocessed Tweet	***POS Tagger* Output**
[@user1, @user2, Oh, my, God, I, just, watched, Michael's, comeback, !, you, remember, him, from, the, 90, s, ?, ?, yeah, ;), #michaelcomeback]	[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, Michael's/Z, comeback/N, !/,, you/O, remember/V, him/O, from/P, the/D, 90/$, s/G, ?/,, ?/,, yeah/!, ;)/E, #michaelcomeback/#]

4.6 Feature Extraction

The feature extraction process is a key component of *SentiStorm*. It is responsible for the predicting quality of the follow-up *Support Vector Machine* component. The *Feature Vector Generation* component extracts numerical features out of the preprocessed and tagged tweets. For that purpose, it uses a rich feature set, which consists of Term Frequency–Inverse Document Frequency (TF–IDF), POS tags and sentiment lexica. The following sub-sections present each component of the feature extraction process in detail.

4.6.1 TF–IDF

Term Frequency–Inverse Document Frequency (TF–IDF) is a weighting scheme based on the bag-of-words model. The bag-of-words model ignores the order of terms and only stores the number of occurrences. In extension, TF–IDF weights the term frequency by the number of occurrences of a term in all documents [MRS08, p. 117].
The term frequency can be defined by multiple different approaches such as [MRS08, pp. 126–128]:

- **Raw term frequency**
  The simplest approach is called raw term frequency, because it is defined by the raw number of occurrences $\mathrm{f}(t, d)$ of term t in document d. It is defined by Equation 4.1.

$$\mathrm{tf}_{\mathrm{raw}}(t, d) = \mathrm{f}(t, d) \tag{4.1}$$

- **Boolean term frequency**
  The boolean term frequency, which is presented in Equation 4.2, can only be 1 or 0. If there are any occurrences of term t in document d, then the boolean term frequency is 1, otherwise it is 0.

$$\mathrm{tf}_{\mathrm{bool}}(t, d) = \begin{cases} 1 & \text{if } \mathrm{f}(t, d) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

- **Logarithmically scaled term frequency**
  Logarithmically scaling is a commonly used approach for the term frequency. It is based on the logarithm of the number of occurrences of term t in document d. If a term t does not occur in a document d at all, then the term frequency is set to 0. Equation 4.3 defines the logarithmically scaled term frequency.

$$\mathrm{tf}_{\log}(t, d) = \begin{cases} 1 + \log \mathrm{f}(t, d) & \text{if } \mathrm{f}(t, d) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The inverse document frequency (IDF) is used as the weighting factor in TF–IDF. It is defined by the logarithm of the relation between the total number of documents N and the document frequency. The document frequency df of term t is the number of documents out of a collection of N documents that contain the term t. The IDF of term t is defined by the following Equation 4.4 [MRS08, p. 118].

$$\mathrm{idf}(t) = \log \frac{N}{\mathrm{df}(t)} \tag{4.4}$$

If a term is often used in a collection of documents, then its IDF is low. Otherwise, if a term is very rare, then the IDF will get high.

The TF–IDF weighting scheme $w_{t,d}$ combines the term frequency (TF) and the inverse document frequency (IDF). The purpose of TF–IDF is to have a combined weight for each term t in each document d. It is defined by Equation 4.5 [MRS08, pp. 118–119].

$$w_{t,d} = \mathrm{tfidf}(t,d) = \mathrm{tf}(t,d) \times \mathrm{idf}(t) \tag{4.5}$$

The n-dimensional TF–IDF feature vector $\mathbf{w_d}$ is defined in Equation 4.6. It consists of the n weights $w_{t,d}$, which are based on n terms and one document d. The set of n terms is composed by the union of all terms in the entire collection. *SentiStorm* removes Twitter-specific terms such as at-mentions, URLs, email addresses and stop words. Stop words are terms, which are so common that they are sentimentally empty and can be omitted.

$$\mathbf{w_d} = (w_{t_1,d}, w_{t_2,d}, \ldots, w_{t_n,d}) \tag{4.6}$$

Equation 4.7 presents the normalization $\hat{\mathbf{w}}$ of the TF–IDF vector $\mathbf{w}$. It is used to scale the feature vector $\mathbf{w}$ to unit-length, because it is recommended to scale the data to a $[0, 1]$ interval for the SVM.

$$\hat{\mathbf{w}}_\mathbf{d} = \frac{\mathbf{w_d}}{\|\mathbf{w_d}\|_2} = \frac{\mathbf{w_d}}{\sqrt{w_{t_1,d}^2 + w_{t_2,d}^2 + \ldots + w_{t_n,d}^2}} \tag{4.7}$$

In *SentiStorm* the TF–IDF feature vector generation is based on the logarithmically scaled term frequency $\mathrm{tf}_{\log}(t,d)$ and the vector normalization of Equation 4.7. According to [MN14, p. 3], the combination of the logarithmically scaled TF and the IDF is the best known weighting scheme in information retrieval. The difference between the logarithmically scaled and raw term frequency is minimal in short tweet texts because usually a tweet does not consist of multiple repeating words.

Table 4.8: Tagged example tweets

Id	Tagged Tweet
d_1	[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, Michael's/Z, comeback/N, !/,, you/O, remember/V, him/O, from/P, the/D, 90/$, s/G, ?/,, ?/,, yeah/!, ;)/E, #michaelcomeback/#]
d_2	[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, it/O, again/R, !/,, ;)/E, #michaelcomeback/#]
d_3	[@user3/@, I/O, love/V, him/O, too/R, !/,, ;)/E, #michaelcomeback/#]

Table 4.8 illustrates three tagged example tweets, which are used in the following TF–IDF feature vector generation example. In the first step, *SentiStorm* creates a set of terms by the union of all terms of each document. Twitter-specific terms such as at-mentions, URLs, email addresses or emoticons are excluded. *SentiStorm* also ignores stop words such as *I*, *you* or *my* as well as numbers and punctuations. The final set of nine terms for the three example documents d_1, d_2 and d_3 looks as follows:

$$\{\text{oh}, \text{god\#n}, \text{watch\#v}, \text{michael'\#n}, \text{comeback\#n},$$
$$\text{remember\#v}, \text{yeah}, \text{michaelcomeback}, \text{love\#v}\}$$

The POS tags of nouns, verbs, adjectives and adverbs are attached to each term. *SentiStorm* also extracts the hashtags and includes them in the term set. Based on this set of terms Table 4.9 presents the TF–IDF calculations of the tagged example tweets of Table 4.8. In a real-world application this table would be huge because the TF–IDF has to be calculated for each term in the training dataset. The base of the logarithm function does not matter, in this example a base of 10 is used.

Table 4.9: TF-IDF of the tagged example tweets

Term t	$\text{tf}_{\log}^{d_1}$	$\text{tf}_{\log}^{d_2}$	$\text{tf}_{\log}^{d_3}$	$\text{df}(t)$	$\text{idf}(t)$	$\hat{\mathbf{w}}_{\mathbf{d_1}}$	$\hat{\mathbf{w}}_{\mathbf{d_2}}$	$\hat{\mathbf{w}}_{\mathbf{d_3}}$
oh	1	1	0	2	0.1761	0.1758	0.5774	0
god#n	1	1	0	2	0.1761	0.1758	0.5774	0
watch#v	1	1	0	2	0.1761	0.1758	0.5774	0
michael'#n	1	0	0	1	0.4771	0.4763	0	0
comeback#n	1	0	0	1	0.4771	0.4763	0	0
remember#v	1	0	0	1	0.4771	0.4763	0	0
yeah	1	0	0	1	0.4771	0.4763	0	0
michaelcomeback	1	1	1	3	0	0	0	0
love#v	0	0	1	1	0.4771	0	0	1

For example, the $\mathrm{tf}_{\log}(\mathrm{oh}, d_1)$ is calculated by $1 + \log(1.0) = 1$, because it occurs only once in document d_1. The $\mathrm{df}(\mathrm{oh})$ is 2, because two documents d_1 and d_2 contain the term *oh*. Based on the total number of documents $N = 3$, the inverse document frequency $\mathrm{idf}(\mathrm{oh})$ is computed by $\log_{10} \frac{3}{2} = 0.1716$. Therefore, the TF–IDF weight of the term *oh* is calculated by $w_{\mathrm{oh},d_1} = \mathrm{tf}(\mathrm{oh}, d_1) \times \mathrm{idf}(\mathrm{oh}) = 0.1716$. Finally, *SentiStorm* normalizes the TF–IDF vectors $\mathbf{w_d}$ with the Euclidean norm, which results in $\|\mathbf{w_{d_1}}\|_2 = 1.0018$ for document d_1, $\|\mathbf{w_{d_2}}\|_2 = 0.305$ for document d_2 and $\|\mathbf{w_{d_3}}\|_2 = 0.4771$ for document d_3. The first element of the final normalized TF–IDF vector $\hat{\mathbf{w}}_{\mathbf{d_1}}$ is computed by $\frac{w_{\mathrm{oh},d_1}}{\|\mathbf{w_{d_1}}\|_2} = 0.1758$.

Table 4.10 illustrates the final normalized TF–IDF feature vectors $\hat{\mathbf{w}}_{\mathbf{d_1}}$, $\hat{\mathbf{w}}_{\mathbf{d_2}}$ and $\hat{\mathbf{w}}_{\mathbf{d_3}}$ for the tagged tweets d_1, d_2 and d_3. The size of these vectors depends on the number of terms in the previously created term set and stays always the same for every document.

Table 4.10: Normalized TF–IDF feature vectors for the tagged example tweets

Id	Tagged Tweet	TF–IDF Vector
d_1	[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, Michael's/Z, comeback/N, !/,, you/O, remember/V, him/O, from/P, the/D, 90/$, s/G, ?/,, ?/,, yeah/!, ;)/E, #michaelcomeback/#]	[0.1758, 0.1758, 0.1758 0.4763, 0.4763, 0.4763, 0.4763, 0, 0]
d_2	[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, it/O, again/R, !/,, ;)/E, #michaelcomeback/#]	[0.5774, 0.5774, 0.5774 0, 0, 0, 0, 0, 0]
d_3	[@user3/@, I/O, love/V, him/O, too/R, !/,, ;)/E, #michaelcomeback/#]	[0, 0, 0 0, 0, 0, 0, 0, 1]

4.6.2 POS Tags

The *Feature Vector Generation* component uses POS tags to express of how many nouns, verbs, adjectives, adverbs or punctuations a tweet consists. These information increase the feature space and also the prediction quality of the SVM, which is explained by the feature ablation in Section 5.1. The POS feature vector consists of the following eight elements:

1. Number of Nouns
2. Number of Verbs
3. Number of Adjectives
4. Number of Adverbs

5. Number of Injections
6. Number of Punctuations
7. Number of Hashtags
8. Number of Emoticons

These elements are normalized by the total number of terms, because it is recommended to scale the data to a $[0, 1]$ interval for the SVM.

Table 4.11 illustrates the example tweet and its corresponding POS feature vector. This example tweet consists of 23 terms, which have been tagged by the *POS Tagger* component. The example tweet includes the following six nouns {God, I, Michael's, comeback, you, him}. Based on the total number of terms, the normalized number of nouns for this tweet is calculated by $\frac{6}{23} = 0.2609$.

Table 4.11: POS feature vector for the example tweet

Tagged Tweet	**POS Feature Vector**
[@user1/@, @user2/@, Oh/!, my/D, God/^, I/O, just/R, watched/V, Michael's/Z, comeback/N, !/,, you/O, remember/V, him/O, from/P, the/D, 90/$, s/G, ?/,, ?/,, yeah/!, ;)/E, #michaelcomeback/#]	[0.2609, 0.0870, 0, 0.0435, 0.0870, 0.1304, 0.0435, 0.0435]

4.6.3 Sentiment Lexica

SentiStorm uses multiple different sentiment lexica to increase the feature set and reflect the sentiment of the tweet in numerical features as good as possible. The sentiment feature vector consists of the following features for each sentiment lexicon:

1. Number of positive sentiments
2. Number of neutral sentiments
3. Number of negative sentiments
4. Sum of the sentiment scores
5. Number of the total sentiments
6. Maximum of the positive sentiment scores
7. Maximum of the negative sentiment scores

Table 4.12 presents the different sentiment lexica, which are used by *SentiStorm*. It also includes the number of terms and the range of the sentiment scores. Each sentiment lexicon consists of a set of tokens, which are assigned by a sentiment score. *SentiStorm* uses only unigram tokens, but a future extension might use bigrams or trigrams as well. Bigrams and trigrams are useful to recognize negations in phrases.

Table 4.12: Sentiment lexica

Sentiment Lexicon	# of Terms	Scores
AFINN-111 [Nie11b] [Nie11a]	2477 words	$[-5, 5]$
SentiStrength Emotions [The15]	2,544 regex	$[-5, 5]$
SentiStrength Emoticons [The15]	107 emoticons	$[-1, 1]$
SentiWords [GGT13] [Tec15]	147,292 words	$[-0.935, 0.88257]$
Sentiment140 [KZM14] [Moh15]	62,468 unigrams	$[-4.999, 5]$
Bing Liu [HL04] [Liu15]	6,785 words	[positive, negative]
MPQA Subjectivity [WWH05] [MPQ15]	6,886 words	[positive, negative]

SentiStorm loads every sentiment lexicon into the memory and uses feature scaling to adjust the sentiment score to a $[0, 1]$ interval. Feature scaling is presented by Equation 4.8. The normalized value $\hat{x}$ depends on the minimum score and the maximum score of all instances in the sentiment lexicon [WFH11, p. 132].

$$\hat{x} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{4.8}$$

For example, the sentiment score -4 of the AFINN lexicon is scaled to $\hat{x} = \frac{-4+5}{5+5} = 0.1$. After feature scaling, every sentiment lexicon uses the same interval for their sentiment scores and therefore can be easily combined. The Bing Liu and MPQA Subjectivity lexica have only positive and negative scores, which are mapped to 1 and 0.

Table 4.13 illustrates the example tweet and its corresponding sentiment feature vector based on the AFINN and SentiWords sentiment lexica.

Table 4.13: AFINN sentiment feature vector for the example tweet

Lexicon	Tagged Tweet	Vector
AFINN-111	[@user1, @user2, Oh, my, **God/0.6**, I, just, watched, Michael's, comeback, !, you, remember, him, from, the, 90, s, ?, ?, **yeah/0.6**, ;), #michaelcomeback]	[2, 0, 0, 1.2, 2.0, 0.6, 0]
SentiWords	[@user1, @user2, **Oh/0.5144**, my, **God/0.6382**, **I/0.5144**, **just/0.6134**, **watched/0.5764**, Michael's, **comeback/0.6713**, !, you, **remember/0.7208**, him, from, the, **90/0.5144**, **s/0.5144**, ?, ?, **yeah/0.5144**, ;), #michaelcomeback]	[5, 5, 0, 5.7922, 10.0, 0.7208, 0]

The example tweet consists of two positive tokens based on the AFINN sentiment lexicon. The terms *God* and *yeah* have both a sentiment score

of 0.6, which results in a total score of 1.2. The feature vector based on the AFINN lexicon shows two positive terms, a sum of 1.2 and a maximum sentiment score of 0.6. In contrast, the SentiWords lexicon recognizes more tokens. Sentiment scores are identified as neutral, when the score is in the interval $[0.45, 0.55]$. The feature vector based on the SentiWords lexicon identified five positive and five neutral tokens. The most positive term is *remember* with a sentiment score of 0.7208. Emoticon tokens are only recognized by the SentiStrength Emoticons lexicon. *SentiStorm* concatenates all feature vectors of each lexicon.

4.7 Support Vector Machine (SVM)

The last component of the *SentiStorm* topology is the *Support Vector Machine* (SVM). SVM classification was proposed by Cortes et al. [CV95] by their approach of Support Vector Networks in 1995. SVM is used to classify the sentiment of a tweet based on its feature vector. It is a supervised learning model and requires a set of training data and associated labels. The training data consist of feature vectors, which are usually defined by numerical values. The feature vector extraction of a tweet has been explained in Section 4.6. The SVM tries to find hyperplanes that separate these training vectors based on their associated labels [CST00, p. 9]. Then all future feature vectors can be classified.
SentiStorm uses the `LIBSVM` library of Chang et al. [CL11]. It is a well-known SVM implementation in the machine learning area. In 2000, Chang et al. started developing the `LIBSVM` library in C++ and it was later translated to many other different programming languages such as Java or Python [CL11, pp. 1–2]. *SentiStorm* uses the Java implementation, which is included in the `LIBSVM` library and was developed by unknown authors at National Taiwan University. This library provides implementations for Support Vector Classification (SVC), Support Vector Regression (SVR) and one-class SVM [CL11, p. 2].

Figure 4.4 illustrates a linearly separable two-class dataset, which is separated by a maximum margin between two hyperplanes. These two hyperplanes are indicated by dashed lines. The process of finding these hyperplanes in a linearly separable two-class dataset is now explained in detail. Afterwards, the process for nonseparable datasets is presented. Both processes are part of the Support Vector Classification (SVC), which is based on a set of n-dimensional training vectors $\mathbf{x}_i, i = 1, \ldots, l$. Each training vector is assigned by a label y_i, which defines the sentiment class of this vector. In a two-class dataset, y_i is an element of $\{-1, 1\}$, which for example is representing a positive or negative sentiment. In Figure 4.4 the red triangles illustrate the training vectors of the class

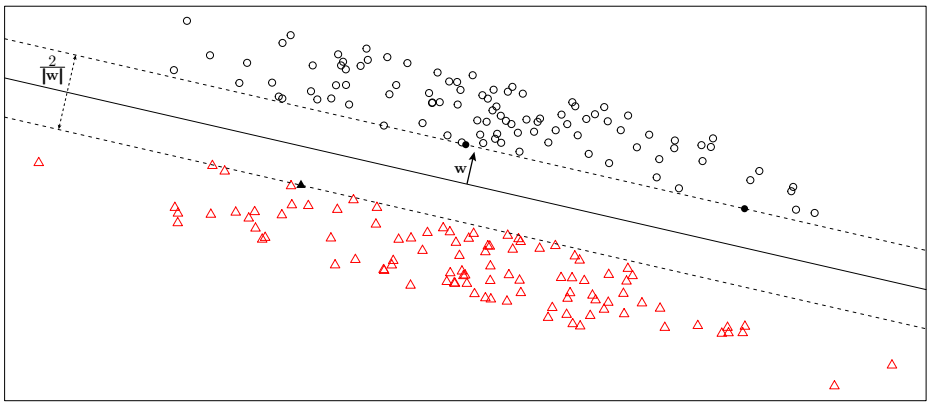

Figure 4.4: Illustration of a linearly separable two-class dataset

$y_i = -1$, the black circles belong to the class $y_i = 1$. The goal of the SVC is to find a hyperplane with a normal vector $\mathbf{w}$, which separates the two-class dataset. A hyperplane consists of a set of points $\mathbf{x}$ and is defined by $\mathbf{w}^T\mathbf{x}+w_0 = 0$. Assuming that the two-class dataset is linearly separable, SVC is able to find two additional hyperplanes, so that all training vectors $\mathbf{x}_i$ satisfy the following inequations [TK08, pp. 118–119]:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + w_0 &\geq 1, \quad \forall \mathbf{x}_i \text{ with } y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + w_0 &\leq -1, \quad \forall \mathbf{x}_i \text{ with } y_i = -1 \end{aligned}$$

These two hyperplanes define the maximum margin between the two classes, with the additional condition that no points lie in between. All training vectors $\mathbf{x}_i$ that are elements of the two hyperplanes are called support vectors. In Figure 4.4 the support vectors are indicated by black triangles and circles, the maximum margin is delimited by the dashed lines. The width of the margin is defined by $\frac{2}{\|\mathbf{w}\|}$. The maximization of this margin is achieved by minimizing $\frac{1}{2}\mathbf{w}^T\mathbf{w}$. The optimization problem can be formulated by the following conditions [CL11, p. 3] [TK08, pp. 119–121]:

$$\begin{aligned} \min_{\mathbf{w},w_0} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1, i = 1, \ldots, l \end{aligned} \tag{4.9}$$

In real-world applications the dataset is often nonseparable. In this case the SVC is not able to find two hyperplanes, which completely separate the classes.

Figure 4.5 illustrates a nonseparable two-class dataset, where now training vectors are also located between the two hyperplanes. These vectors can be categorized in correctly classified but lying within the margin

and misclassified. Misclassified training vectors are marked by a square in Figure 4.5.

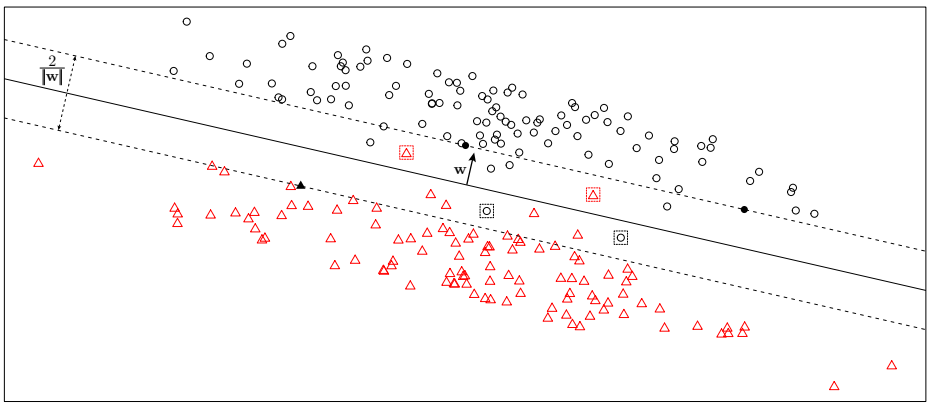

Figure 4.5: Illustration of a nonseparable two-class dataset

The C-Support Vector Classification (C-SVC) solves the problem of nonseparable datasets by introducing a new parameter C and the variable ξ. The C-SVC optimization problem is updated and formulated as follows [TK08, pp. 124–126]:

$$\begin{aligned} \min_{\mathbf{w},w_0,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=0}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{4.10}$$

The variable ξ is also called slack variable, because it is used to allow training vectors to lie between the two hyperplanes. The goal of C-SVC is still to find the maximum margin between the two hyperplanes, but now also to keep the number of vectors lying within this margin as small as possible. For these vectors is $\xi > 0$. The additional parameter C indicates a penalty value for all vector lying within the margin. If the penalty weight of C is chosen very high, then the margin shrinks [TK08, p. 125]. The balance between the parameter C and the maximum margin can be obtained by a parameter grid search, which is explained in Section 4.7.1.
The explanations above are based on a linear classifier. But SVC is also capable of using non-linear classification, which is realized by kernel functions. `LIBSVM` supports the following four kernel functions [HCL03, p. 2]:

1. Linear kernel function
   $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$

2. Polynomial kernel function
   $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \gamma > 0$
3. Radial Basis Function (RBF)
   $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$
4. Sigmoid kernel function
   $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$

SentiStorm uses the C-SVC implementation of `LIBSVM` together with the radial basis kernel function. RBF is the default kernel function of `LIBSVM`, because it has multiple advantages compared to the linear, polynomial and sigmoid kernel functions [HCL03, p. 4]. Furthermore, the RBF achieves a good performance on the SemEval 2013 dataset and therefore, *SentiStorm* sticks to this kernel function. The parameter search for values of C and γ, which are used in the RBF, is explained in the following section.

4.7.1 Grid Search

Since *SentiStorm* uses C-SVC together with the RBF kernel, the parameters C and γ have to be specified. The goal of the grid search is to find the optimal values for these parameters. They are not known beforehand and have to be determined for every new dataset. Since the evaluations of *SentiStorm* are based on the SemEval 2013 dataset, the grid search is also processed for this dataset. The parameter search uses an n-fold cross-validation to find the maximum accuracy of different parameter values. The n-fold cross-validation divides the training vectors into n subsets of equal size. Then each subset is processed sequentially. Based on the $n - 1$ remaining subsets a classifier is built and tested on the actual subset. The total cross-validation accuracy is calculated by average cross-validation results of each subset. The accuracy indicates the percentage of correctly predicted vectors [HCL03, p. 5].

A grid search is processed in two steps. In the first step a coarse-grained search is conducted based on exponentially growing sequences of C and γ. Due to the recommendations in [HCL03, p. 5], the coarse-grained grid search uses the following values of parameter C and γ:

$$C \in \{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$$
$$\gamma \in \{2^{-15}, 2^{-13}, \ldots, 2^{1}, 2^{3}\}$$

In the second step a fine-grained grid search is conducted. It is based on the n-fold cross validation accuracies of the coarse-grained search. The values of C and γ are now based on a subset of the coarse-grained sequence, which has produced the best n-fold cross-validation accuracies.

Figure 4.6 illustrates the coarse-grained grid search results, which are based on a 10-fold cross-validation of the SemEval 2013 dataset. The highlighted rectangle indicates the area with the best cross-validation accuracies. In this case, the C parameter is between 2^6 and 2^{12} and the γ is in the range of 2^{-8} to 2^{-14}. The best accuracy of 70.59% has been discovered with the parameters $C = 2^{11}$ and $\gamma = 2^{-13}$.

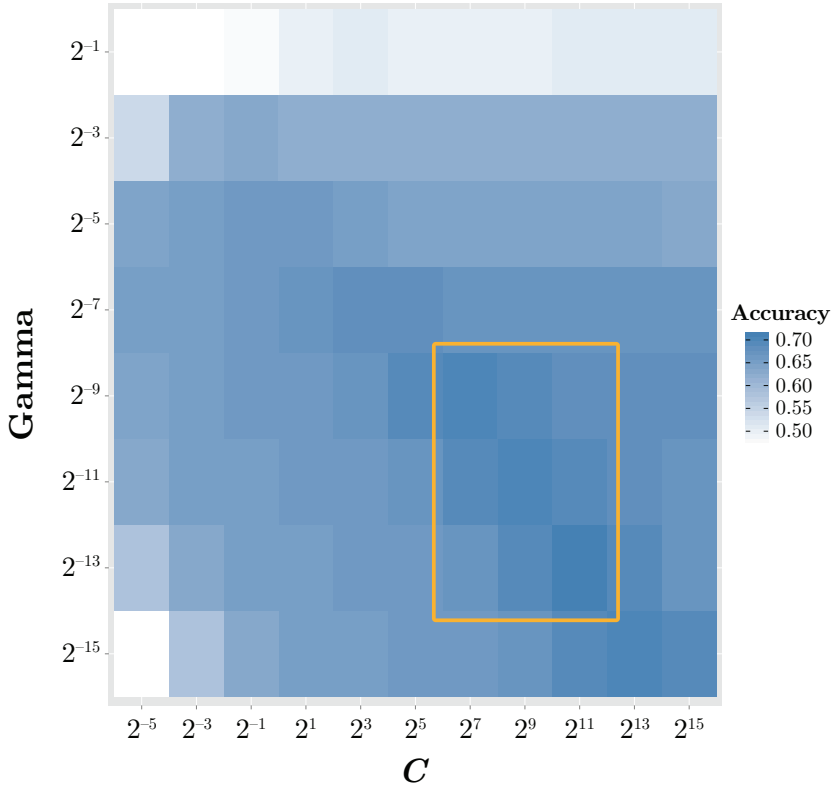

Figure 4.6: Coarse-grained grid search based on SemEval 2013

Figure 4.7 illustrates the fine-grained grid search. The darkest box in the middle indicates the highest cross-validation accuracy of 70.71%, which is measured with $C = 2^9$ and $\gamma = 2^{-11}$. These parameters are used by *SentiStorm* to achieve the highest possible accuracy.

4.7.2 Class Weights

Class weights are used to balance unequal distributed training vectors of several different classes. These weights are very important for unbalanced classes in SVC. In this section the class weights are specified for the SemEval 2013 dataset, because all evaluations of *SentiStorm* are based on this dataset.

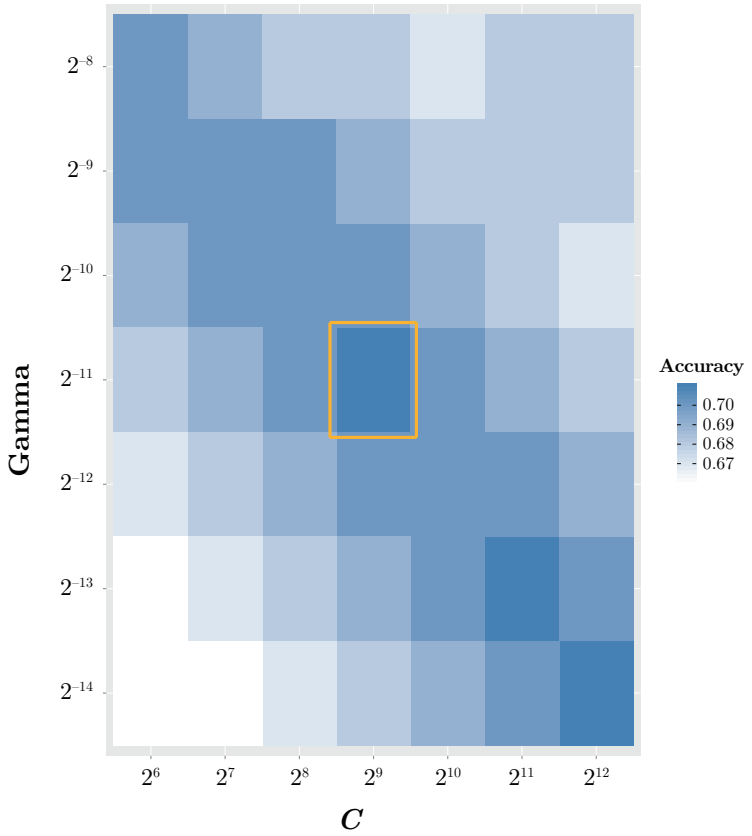

Figure 4.7: Fine-grained grid search based on SemEval 2013

Table 4.14 illustrates the training dataset statistics of SemEval 2013 and the corresponding class weights. *SentiStorm* uses the train and dev dataset of SemEval 2013 for training the C-SVC. This includes 11,382 training vectors, which are grouped into three classes positive, negative and neutral. Since the number of neutral tweets is almost three times higher than the negative tweets, the classes have to be weighted. The weight of the neutral class is set to one and based on this, the positive class is weighted with 1.26 and the negative class with 2.96.

Table 4.14: SemEval 2013 class weights [NRK+13, p. 314]

	Positive	**Negative**	**Neutral Objective**	**Total**
Twitter2013 train	3660	1466	4602	9728
Twitter2013 dev	575	340	739	1654
Total	4235	1806	5341	11382
Weights	**1.26**	**2.96**	**1.0**	

Chapter 5

SentiStorm Evaluation

This chapter presents the evaluation of *SentiStorm* and is divided into a quality evaluation and a performance evaluation. The quality evaluation compares the sentiment prediction quality of *SentiStorm* with state-of-art sentiment classification systems based on the SemEval 2013 dataset. The performance evaluation discusses the speed of *SentiStorm* on a single node without Storm and on a multi-node Storm cluster. The stand-alone performance is evaluated based on the GATE and the ARK POS tagger. Finally, this chapter shows, that the performance of *SentiStorm* on a multi-node Storm cluster meets the real-time requirement.

5.1 Quality Evaluation

The quality evaluation is based on the SemEval dataset of 2013. The statistics of this dataset are illustrated in Table 5.1. The train and development datasets are used for training the SVM classifier. The predominant number of tweets are assigned by a neutral class label. Therefore, the class weights, which are presented in Section 4.7.2, compensate this unbalanced dataset. The test dataset is used as input for *SentiStorm*. The output of *SentiStorm* is a sentiment prediction for each tweet, which can be compared to the original class label of the tweet.

Table 5.1: SemEval 2013 dataset statistics [NRK+13, p. 314]

	Positive	**Negative**	**Neutral Objective**	**Total**
Twitter 2013 train	3660	1466	4602	9728
Twitter 2013 dev	575	340	739	1654
Twitter 2013 test	1572	601	1640	3813

In a quality evaluation it is important to consider wrong predictions. Therefore, a confusion matrix is used to reflect wrong classifications for

each class [WFH11, p. 163].
Table 5.2 illustrates a three-class confusion matrix. A confusion matrix $M = [M(i,j)]$ represents the numbers of instances, which were originally assigned to the class label i and have been predicted to the class label j. The correctly classified instances are located in the diagonal entries $M(i,i)$. Every other entry represents wrongly predicted instances [TK08, p. 573].

Table 5.2: Three-Class Confusion Matrix based on [WFH11, p. 164]

		Predicted Class		
		Positive	Negative	Neutral
Actual Class	Positive	true positive	false negative	false neutral
	Negative	false positive	true negative	false neutral
	Neutral	false positive	false negative	true neutral

Based on the confusion matrix the following measurements are used for the quality evaluation of *SentiStorm* and the comparison with other state-of-art sentiment classification systems [TK08, p. 573]:

- **Accuracy**
  The overall accuracy represents the percentage of correctly predicted instances. The percentage is the proportion between the correctly predicted instances and the total number of instances in the test set. The accuracy A for n classes relies on the confusion matrix M and can be calculated by Equation 5.1.

$$A = \frac{\sum_{i=1}^{n} M(i,i)}{\sum_{i,j=1}^{n} M(i,j)} \tag{5.1}$$

- **Recall**
  Recall R_i is the percentage of instances of class i, which have been correctly predicted. The recall measurement R_i is defined by Equation 5.2.

$$R_i = \frac{M(i,i)}{\sum_{j=1}^{n} M(i,j)}, \quad 1 \leq i \leq n \tag{5.2}$$

- **Precision**
  Precision P_i is the percentage of instances, which have been predicted as class i and really belong to the actual class i. Equation 5.3 illustrates the precision P_i.

$$P_i = \frac{M(i,i)}{\sum_{j=1}^{n} M(j,i)}, \quad 1 \leq i \leq n \tag{5.3}$$

The F-score or F-measure combines the recall R_i and precision P_i. It is the harmonic mean of recall and precision of class i. Its best value

is denoted by 1 and the worst value by 0. Equation 5.4 defines the F-measure [WFH11, p. 175] [NRK+13, p. 315].

$$F_i = 2 \cdot \frac{P_i \cdot R_i}{P_i + R_i} \tag{5.4}$$

SemEval uses the average of the F-measures of the positive and negative class to compare the quality of sentiment classification systems. Equation 5.4 illustrates this average positive and negative F-measure [NRK+13, p. 315].

$$F_{p/n} = \frac{F_{pos} + F_{neg}}{2} \tag{5.5}$$

The F_{all}-measure is the average F-measure of every class i weighted by the total number of instances w_i, which have been classified to this class i. Equation 5.6 illustrates the F_{all}-measure.

$$F_{all} = \frac{\sum_{i=1}^{n} F_i \cdot w_i}{\sum_{i=1}^{n} w_i} \tag{5.6}$$

For example, a three-class F_{all}-measure consisting of the classes positive, negative and neutral is shown by Equation 5.7.

$$F_{all} = \frac{F_{pos} \cdot w_{pos} + F_{neg} \cdot w_{neg} + F_{ntr} \cdot w_{ntr}}{w_{pos} + w_{neg} + w_{ntr}} \tag{5.7}$$

Table 5.3 illustrates the confusion matrix of *SentiStorm* based on the SemEval 2013 test dataset. *SentiStorm* classified 2677 tweets correctly, which is an overall accuracy of 70.21%. This accuracy is calculated by the following equation:

$$A = \frac{\sum_{i=1}^{n} M(i,i)}{\sum_{i,j=1}^{n} M(i,j)} = \frac{1033 + 412 + 1232}{3813} = 0.7021$$

Table 5.3: *SentiStorm* Confusion Matrix

		Predicted Class			
		Positive	Negative	Neutral	Total
Actual Class	Positive	1033	146	393	1572
	Negative	56	412	133	601
	Neutral	257	151	1232	1640
	Total	1346	709	1758	3813

Based on the confusion matrix of Table 5.3, the quality of *SentiStorm* can be further evaluated. First the recall and precision is determined for each class. Then the F-measure is calculated based on these values. Finally, the weighted F-measures are used to compare the quality.

Table 5.4: *SentiStorm* Quality Results

Class i	**Recall R_i**	**Precision P_i**	**F_i-measure**
Positive	$\frac{1033}{1572} = 0.6571$	$\frac{1033}{1346} = 0.7675$	$2 \cdot \frac{0.7675 \cdot 0.6571}{0.7675+0.6571} = 0.7080$
Negative	$\frac{412}{601} = 0.6855$	$\frac{412}{709} = 0.5811$	$2 \cdot \frac{0.5811 \cdot 0.6855}{0.5811+0.6855} = 0.6290$
Neutral	$\frac{1232}{1640} = 0.7512$	$\frac{1232}{1758} = 0.7008$	$2 \cdot \frac{0.7008 \cdot 0.7512}{0.7008+0.7512} = 0.7251$

Table 5.4 illustrates the recall, precision and F-measure for each class. Based on these results the weighted $F_{p/n}$-measure and F_{all}-measure can be calculated as follows:

$$F_{p/n} = \frac{F_{pos} + F_{neg}}{2} = \frac{0.7080 + 0.6290}{2} = 0.6685$$

$$F_{all} = \frac{F_{pos} \cdot w_{pos} + F_{neg} \cdot w_{neg} + F_{ntr} \cdot w_{ntr}}{w_{pos} + w_{neg} + w_{ntr}} =$$

$$= \frac{0.7080 \cdot 1346 + 0.6290 \cdot 709 + 0.7251 \cdot 1758}{1346 + 709 + 1758} = 0.7012$$

The $F_{p/n}$-measure of *SentiStorm* is 66.85%, which would achieve the second place in the top five SemEval message polarity results of 2013. Table 5.5 shows the top five SemEval results of 2013 and 2014 based on the test dataset of 2013. In 2014, the quality $F_{p/n}$-measure increased to over 70% based on the same test dataset. For example, the leading team NRC-Canada improved their sentiment classification system from 69.02% to 70.75% by primarily increasing the sentiment lexica. But the quality of *SentiStorm* is also comparable to the top ten results of the 2014 [NRK+13, p. 318] [RRNS14, p. 78].

Table 5.5: Top Five SemEval Message Polarity Results

(a) 2013 [NRK+13, p. 318]

Team	**$F_{p/n}$**
NRC-Canada	0.6902
GU-MLT-LT	0.6527
teragram	0.6486
BOUNCE	0.6353
KLUE	0.6306

(b) 2014 [RRNS14, p. 78]

Team	**$F_{p/n}$**
TeamX	0.7212
NRC-Canada	0.7075
coooolll	0.7040
RTRGO	0.6910
SentiKLUE	0.6906

The feature ablation of Table 5.6 illustrates how much impact different features have on the overall prediction quality. The RBF kernel function is used together with the parameter $C = 2^9$ and $\gamma = 2^{-11}$, which were obtained by the grid search earlier. Also the class weights of 1.26 for positive, 2.96 for negative and 1 for neutral are set to balance the training dataset. The first evaluation is based on a 10-fold cross validation of the SemEval 2013 train and dev dataset. The second results are measured by the SemEval 2013 test dataset, which are based on the confusion matrix of Table 5.3. Each row presents F-measures, which are obtained by subtracting one feature from all features. The difference measurements $D_{F_{all}}$ and $D_{F_{p/n}}$ show the impact of the removed feature. The most important features are the class weights and TF–IDF, which improve the $F_{p/n}$-measure by 0.0354 and 0.0287. The sentiment lexica of Bing Liu and MPQA have only a minimal impact in the prediction quality.

Table 5.6: *SentiStorm* Feature Ablation

	SemEval 2013 Train and Dev						**SemEval 2013 Test**							
	F_{pos}	F_{neg}	F_{ntr}	F_{all}	$F_{p/n}$	**A**	F_{pos}	F_{neg}	F_{ntr}	F_{all}	$F_{p/n}$	**A**	$D_{F_{all}}$	$D_{F_{p/n}}$
All Features	.7099	.5663	.7406	.7067	.6381	.7041	.7080	.6290	.7251	.7012	.6685	.7021		
- Class Weights	.7111	.5556	.7459	.7105	.6333	.7066	.7021	.5642	.7302	.7023	.6331	.6974	+.0011	-.0354
- TF-IDF	.6793	.5272	.7157	.6790	.6032	.6756	.6380	.7354	.6689	.6634	.6398	.6666	-.0378	-.0287
- POS Tags	.7073	.5616	.7407	.7053	.6345	.7026	.7049	.6014	.7148	.6903	.6531	.6916	-.0109	-.0154
- AFINN	.7035	.5514	.7367	.7013	.6274	.6981	.6952	.6138	.7082	.6857	.6545	.6869	-.0155	-.0140
- SentiStren[1]	.7108	.5598	.7391	.7055	.6353	.7028	.7070	.6218	.7247	.6993	.6644	.7002	-.0019	-.0041
- SentiStren :-)[2]	.6985	.5544	.7379	.6997	.6265	.6969	.6938	.6138	.7180	.6905	.6538	.6910	-.0107	-.0147
- SentiWords	.7108	.5640	.7420	.7073	.6374	.7047	.7003	.6094	.7246	.6951	.6549	.6958	-.0061	-.0136
- Sentiment140	.7036	.5633	.7398	.7042	.6335	.7013	.6972	.6051	.7222	.6918	.6511	.6926	-.0094	-.0174
- Bing Liu	.7077	.5620	.7378	.7041	.6348	.7014	.7031	.6261	.7242	.6989	.6646	.6994	-.0023	-.0039
- MPQA	.7084	.5641	.7394	.7052	.6363	.7026	.7075	.6159	.7279	.7002	.6617	.7010	-.0010	-.0068

[1]SentiStrength Emotions lexicon
[2]SentiStrength Emoticons lexicon

5.2 Performance Evaluation

The performance evaluation analyzes the speed of *SentiStorm*. The speed is mostly measured in tuples per second, which in this case are tweets per second. In a first step the stand-alone performance of *SentiStorm* is evaluated, which is conducted for different numbers of threads. Then the performance of *SentiStorm* on a Storm cluster is presented. The performance evaluations are based on Amazon *c3.8xlarge* EC2 instances. The specification of these instances is illustrated in Table 5.7.

Table 5.7: Amazon *c3.8xlarge* EC2 instance specification

CPU	Intel Xeon E5-2680 v2 (Ivy Bridge) 32 cores
Memory	60 GB
SSD	2 x 320 GB
Network	10 Gigabit
OS	ami-b7a130c0 Ubuntu 14.04.1 LTS 64bit (hvm-ssd)
Cost	$1.680 hourly

Since the components of *SentiStorm* are very CPU intensive, the *c3.8xlarge* instance provides 32 cores. Especially the *POS Tagger* and the *SVM* components require a lot of CPU cores. Ubuntu is a well-known Linux operating system and is used on these instances. The operating cost of one *c3.8xlarge* instance is specified by $1.680 per hour.

5.2.1 Stand-alone Performance

The stand-alone performance of *SentiStorm* is evaluated based on different numbers of threads to fully utilize the 32 cores of a single *c3.8xlarge* instance. The topology of *SentiStorm* and its components are executed in a serial fashion. 38130 tweets, which is 10 times the size of the SemEval 2013 test dataset, are used as input. This input dataset is equally distributed among multiple threads. Finally, the number of tweets per second is calculated by the total execution time divided by the total number of input tweets.

Figure 5.1 illustrates the stand-alone performance of *SentiStorm* depending on the ARK and GATE *POS Tagger*. The serial execution on a single thread results in 270 tweets per second based on the ARK tagger and 206 tweets per second based on the GATE tagger. Therefore, the *SentiStorm* implementation based on the ARK tagger is more than 30% faster. The performance increase is pretty impressive until 22 threads. At 22 threads, the ARK tagger implementation achieves 3711 tweets per

second, which is 23% higher than the GATE tagger with 3023 tweets per second. The performance increase stagnates at around 3800 tweets per second for the ARK tagger implementation and at around 3200 tweets per second for the GATE tagger. Therefore, the stand-alone performance of *SentiStorm* on a single node is limited by 3800 tweets per second based on a *c3.8xlarge* EC2 instance.

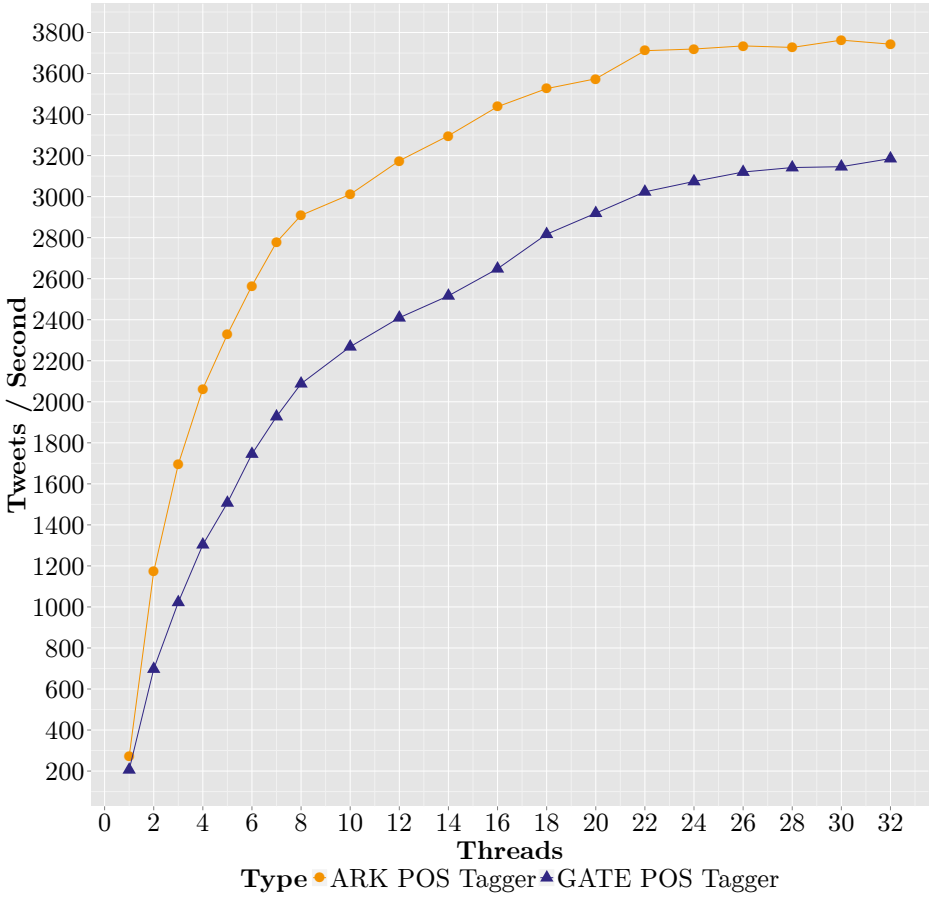

Figure 5.1: Stand-alone performance based on the SemEval 2013 dataset and a *c3.8xlarge* EC2 instance

5.2.2 Storm Performance

The Storm performance evaluation uses a multi-node Storm cluster to scale the performance among multiple *c3.8xlarge* EC2 instances and meet the real-time requirement. This multi-node cluster consists of a single worker per node and goes up to 10 nodes.

Figure 5.2 shows the topology of *SentiStorm* and the corresponding parallelism count of each component. The parallelism value depends on the number of workers or nodes n. For example, the parallelism value of the *POS Tagger* component is 50 for a 10-node cluster, which means that each node executes 5 threads. These parallelism values fully utilize the 32 cores of a *c3.8xlarge* instance, because the *SVM* uses multiple threads too.

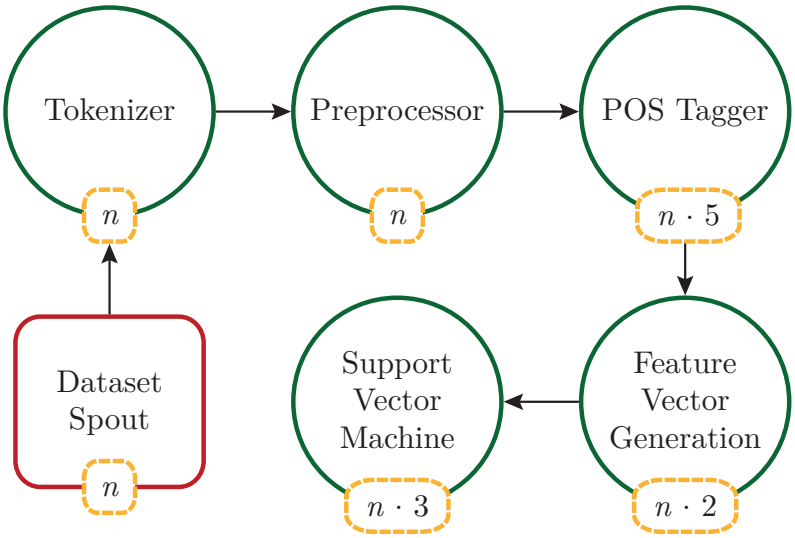

Figure 5.2: Topology of *SentiStorm* including Parallelism

Since the *POS Tagger* and the *SVM* components are the bottleneck of this topology, a higher parallelism count should compensate this. But due to the CPU intensive components and limited number of CPU cores, the bottleneck remains. This could only be solved by decoupling the topology or using a custom scheduler in a future work.

Table 5.8 presents the throughput of *SentiStorm*. The throughput is measured in tweets per second at the end of the topology. The last component of *SentiStorm* is the *SVM*, which actually predicts the sentiment class of a tweet. Each node executes three threads of the *SVM* component based on its parallelism value. The average number of tweets per second decreases only minimal from 1044 tweets per second at one node to 929 tweets per second at 10 nodes. This means that a single-node Storm cluster is able to execute 3133 tweets per second, which is only 20% less than the stand-alone performance. Based on Storm the *SentiStorm* topology scales almost linear and achieves 27,876 tweets per second at 10 nodes. These are 1,672,560 tweets per minute, 100,353,600 tweets per hour and 2,408,486,400 tweets per day. In August 2013 the official number of tweets per second was on average 5,700 and a max-

imum peak value of 143,199 [Twi13]. According to [Int15] the number of tweets per second increased to about 8,793 in 2015. A *SentiStorm* topology consisting of 4 nodes could process this amount of tweets in real-time. Only peak values have to be cached. Therefore, *SentiStorm* is currently able to predict the sentiment of each tweet of the global Twitter stream in real-time.

Table 5.8: *SentiStorm* throughput based on the SemEval 2013 dataset and *c3.8xlarge* EC2 instances

Nodes	**Number of SVM Tasks**	**Average Tweets per SVM Task per Second**	**Average Total Tweets per Second**
1	3	1044	3133
2	6	986	5920
3	9	955	8599
4	12	960	11528
5	15	953	14295
6	18	945	17025
7	21	939	19735
8	24	940	22576
9	27	933	25207
10	30	929	27876

Table 5.9 illustrates the latency of each *SentiStorm* component and the complete latency of the topology. These latencies depend on the different parallelism values for each component, which have been explained above. The *Preprocessor* has the lowest latency of about 0.108 ms. The *POS Tagger* component has the highest latency. It needs about 1.53 ms to process one tweet, which is more than 10 times slower than the *Preprocessor*. *SVM* is slightly faster with a latency of 1.025 ms. The table also shows only a minimal increase in latency for multiple nodes. The complete latency of the topology is about 53.5 ms, which means that it takes 53.5 ms to process a tweet throughout the complete topology. The topology of *SentiStorm* was optimized for high throughput, accepting a higher latency. The tradeoff between latency and throughput depends on how much tuples are allowed to be in flight. A high number of tuples within the topology also means higher queue sizes at each component and a higher latency. In this case the throughput of *SentiStorm* is more important than its latency. In this particular case the best throughput has been observed by setting the number of tuples in flight to 150. If there are more tuples in the topology, a *Spout* will throttle emitting tuples. A higher number of tuples in flight would overflow the queues and Storm would start replaying them.

Table 5.9: *SentiStorm* latencies based on the SemEval 2013 dataset and *c3.8xlarge* EC2 instances

Nodes	**Tokenizer Latency (ms)**	**Preprocessor Latency (ms)**	**POS Tagger Latency (ms)**	**Feature Generation Latency (ms)**	**SVM Latency (ms)**	**Complete Latency (ms)**
1	0.179	0.108	1.492	0.185	0.953	48.155
2	0.182	0.108	1.514	0.183	0.987	51.048
3	0.189	0.112	1.531	0.183	1.034	52.607
4	0.187	0.109	1.543	0.180	1.023	52.311
5	0.188	0.110	1.536	0.183	1.023	52.657
6	0.184	0.108	1.532	0.179	1.025	53.332
7	0.182	0.109	1.544	0.178	1.022	53.575
8	0.187	0.110	1.549	0.178	1.025	53.359
9	0.180	0.107	1.521	0.177	1.016	54.055
10	0.182	0.107	1.528	0.176	1.031	53.889

Figure 5.3 illustrates the throughput of *SentiStorm* based on the different number of nodes. The throughput scales almost linear from a single node to 10 nodes. A single node is able to process 3133 tweets per second compared to 27876 on 10 nodes. It is easily possible to increase the throughput further by adding new nodes.

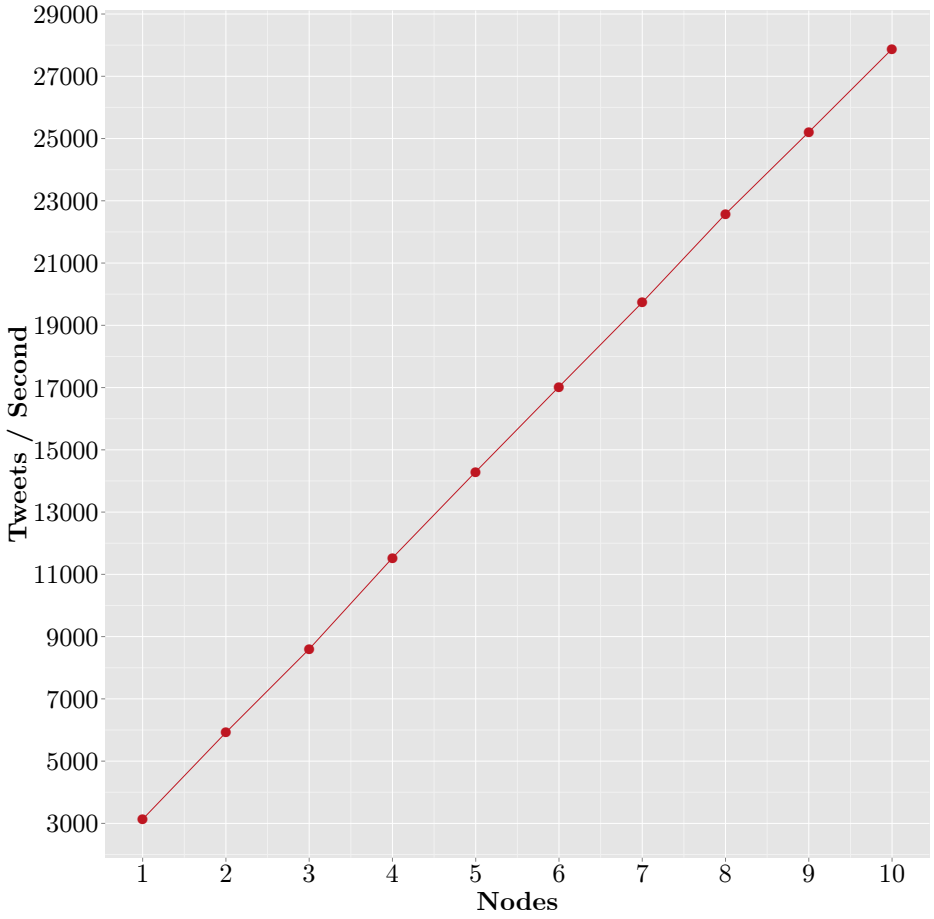

Figure 5.3: *SentiStorm* throughput based on the SemEval 2013 dataset and *c3.8xlarge* EC2 nodes

Chapter 6

Related Work

This chapter presents related work in the field of sentiment analysis and the combination of sentiment analysis with the real-time processing system Apache Storm. The ongoing research in the field of sentiment analysis is enormous, because of the explosion of opinionated user-generated content in social media, twitter, reviews, blogs and much more.
In 2008, Pang and Lee [PL08] presented a survey over the existing techniques and approaches of sentiment polarity detection and subjectivity detection. In 2012, Bing Liu [Liu12] published a book, which contains an up-to-date review of the ongoing research in sentiment analysis. It also includes topics such as document sentiment classification, subjectivity classification and aspect-based sentiment analysis.
The following section presents related work, which is based on the literature surveys of Pang and Lee [PL08] and Bing Liu [Liu12]. In addition to these two reviews it also contains related work of the Semantic Evaluation (SemEval) [Gro15d] exercises, which heavily influenced this master's thesis. Finally, the last section will give an overview of the related work in the field of real-time sentiment analysis based on Apache Storm, which corresponds to the actual research area of this master's thesis.

6.1 Sentiment Analysis

According to Pang and Lee [PL08, p. 6] the term *sentiment analysis* was first mentioned by Das and Chen [DC01] and Tong [Ton01] in 2001. Das and Chen already used different classification algorithms such as Naive classifier and Bayesian classifier to predict the sentiment in stock market web forums. In 2002 Turney [Tur02] used the term *semantic orientation* for an unsupervised classification of reviews. Also in 2002, Pang et al. [PLV02] presented a supervised sentiment classification approach for movie reviews. Since that time, *sentiment analysis* is a well-known

term in the natural language processing research area.
Sentiment analysis has been applied across many different domains such as movie reviews, news, blogs or forums. The following related work focuses on sentiment analysis of tweets because this is the domain what this master's thesis is about. Twitter has become an important source for sentiment analysis because of the availability of huge opinionated user-generated data.
The most important papers related to sentiment analysis on Twitter data in chronological order are Go et al. (2009) [GBH09], Pak and Paroubek (2010) [PP10], Barbosa and Feng (2010) [BF10], Agarwal et al. (2011) [AXV+11], Kouloumpis et al. (2011) [KWM11], Saif et al. (2012) [SHA12] and Dilrukshi et al. (2013) [DDZC13].
Go et al. [GBH09] use supervised classification models such as Naive Bayes, Maximum Entropy (MaxEnt) and Support Vector Machines (SVM). Their feature space consists of unigrams, bigrams and POS tags. The best results are reported with unigrams and SVM at 82.2% accuracy for two classes positive and negative. MaxEnt provided the best accuracy for unigrams and bigrams, while POS tags did not lead to any benefit at all. In contrast to Go et al., Pak and Paroubek [PP10] differentiate between subjective and objective tweets and use three classes positive, negative and neutral. They implemented their own classifier, which is based on the Bayes classifier. The best results were measured with bigrams. Barbosa and Feng [BF10] also use three classes but they present a two-step sentiment analysis approach. In the first step they distinguish between subjective and objective tweets by using a subjectivity classifier. In the second step they categorize the subjective tweets by a polarity classifier. SVM is used as a classifier together with unigrams. In addition to the previously presented approaches they use syntax features such as the number of retweets, hashtags, links, punctuations and exclamation marks, which enhances the accuracy by 2.2% compared to unigrams only. Agarwal et al. [AXV+11] use only two classes and a tree representation of tweets to combine many categories of features. They also use a SVM classifier. The approach of Kouloumpis et al. [KWM11] includes extended features such as emoticons, abbreviations and many more. Their best results were achieved by n-grams and the extended features, while POS tags decreased the performance. Saif et al. [SHA12] introduced a new set of features called semantic features. These semantic features are generated for each entity in a tweet. For example, to the entity "iPhone" the semantic feature "Apple product" is added. Semantic features lead to a performance increase based on a two-class Naive Bayes classifier.
Semantic Evaluation (SemEval) [Gro15d] is an ongoing series of evaluation exercises for the semantic analysis of text, which is organized by the umbrella organization SIGLEX. The main goal of these exercises

is to repeatedly evaluate semantic analysis systems [Gro15d]. The most important publications related to this master's thesis are the approaches of team NRC-Canada Saif et al. [MKZ13] and Zhu et al. [ZKM14], the *KLUE* approach of Proisl et al. [PGEK13] and the *SemantiKLUE* of Evert et al. [EPGK14].
In 2013, the SemEval Task 2 Subtask B [NRK+13, p. 313] covered the topic *Message Polarity Classification* based on tweets. The winner of the SemEval 2013 contest was team NRC-Canada [MKZ13] with a $F_{p/n}$-measure of 0.6902. They used a SVM classifier and multiple sentiment lexica. Their feature space consists of word and character n-grams, POS tags, syntax features, sentiment lexica, emoticons and negation. The *KLUE* approach of Proisl et al. [PGEK13] evaluated multiple classifiers such as Naive Bayes, Linear SVM, and Maximum Entropy. The best result with a $F_{p/n}$-measure of 0.6306 were achieved with the MaxEnt classifier including unigrams and bigrams.
In 2014, the SemEval message polarity classification exercise of 2013 was repeated in Task 9 Subtask B [RRNS14]. The training and development dataset was reused from the Task 2 in 2013, but a new test set was created. The NRC-Canada [ZKM14] improved their F-measure from 0.6902 to 0.7075 based on the same test set of 2013. With the new test set of 2014 the NRC-Canada team achieved the fourth place with a $F_{p/n}$-measure of 0.6985. The extension of the *KLUE* approach now called *SemantiKLUE* of Evert et al. [EPGK14] increased their result from 0.6306 to 0.6906 for the test set of 2013 and achieved a $F_{p/n}$-measure of 0.6702 for the test set of 2014.
The NRC-Canada [ZKM14] improved their approach by using other sentiment lexica and updating their negation handling but they still stick to the SVM classifier. The *SemantiKLUE* approach [EPGK14] also extends the feature set of *KLUE* but reuses the MaxEnt classifier.

6.2 Real-time Sentiment Analysis on Storm

Currently, the research in real-time sentiment analysis especially based on *Apache Storm* is pretty rare, because Storm is a relatively new project. In September 2014, it came into focus when it graduated to an Apache top-level project. A second reason for the small amount of publications might be the lack of interest of researchers in the field of sentiment analysis in real-time capabilities. For these reasons there are only two publications by Artola et al. [ABS14] and Raina et al. [RGS+14] available. Beside these papers there exist several approaches, which include only an unsupervised sentiment analysis based on sentiment lexica. For example, Prashanth Babu [Bab15] used the AFINN [Nie11b] sentiment lexicon to calculate the sentiment of a tweet.

Artola et al. [ABS14] presented a stream processing approach, which consists of a tokenizer, a part of speech tagger (POS), a named entity recognition and classification module (NERC), and a word-sense disambiguation module (WSD). Since 96% of the execution time were spent in the WSD, they increased the parallelism of the WSD module to six instances, which resulted in 296 words per second. The main goal of their Storm pipeline was to convert text documents into NLP Annotation Format (NAF) documents. These documents consist of multiple NLP features such as POS tags, name entities and dependencies, but they actually did not calculate the sentiment for a document.
The only directly related work, which corresponds to this master's thesis, was published by Raina et al. [RGS+14] in November 2014. They present an unsupervised machine learning approach for the sentiment analysis on Apache Storm. The polarity of a tweet is calculated by the sentiment of each word, which was obtained by a sentiment lexicon. They do not use any supervised classification as it is presented in this master's thesis.
Summing up, this master's thesis presents a real-time supervised sentiment classification approach based on Apache Storm, which is currently not presented in any other related work.

Chapter 7

Conclusions and Future Work

This master's thesis gave an introduction of *Apache Storm* and its distributed real-time computation system. It presented related work and commonly used techniques in the field of sentiment analysis.
The main goal of this master's thesis was to implement a real-time sentiment classification system. Therefore, it introduced an approach called *SentiStorm*, which is based on *Apache Storm*. The topology of *SentiStorm* consists of the following five major components:

1. *Tokenizer*
2. *Preprocessor*
3. *POS Tagger*
4. *Feature Vector Generation*
5. *Support Vector Machine (SVM)*

Each component was explained in detail. For example, the *Feature Vector Generation* extracts numerical features out of a tweet text based on TF–IDF, POS tags and multiple sentiment lexica. The extracted feature vectors are processed by the *Support Vector Machine (SVM)*, which predicts the sentiment class based on a training dataset.
Finally, the quality and throughput of *SentiStorm* were discussed in detail. The quality evaluations have shown that *SentiStorm* is comparable with state-of-art sentiment classification systems based on the SemEval dataset of 2013. For example, it achieved a $F_{p/n}$-measure of 66.85%, which would come the second place at the SemEval message polarity results of 2013. But in addition to this high sentiment prediction quality, the performance of *SentiStorm* also showed that a real-time execution is possible. *SentiStorm* is able to process the full Twitter stream in real-time, which is around 8,793 tweets per second. Furthermore, a 10-node cluster is able to process 27,876 tweets per second.

The future work might include improvements in quality and performance. The prediction quality can be improved by using n-grams. N-grams will help recognizing a negation within a phrase. A negation heuristic would be very important. It might be also useful to include n-grams sentiment lexica. Another quality improvement is based on the use of a Word Sense Disambiguation (WSD) algorithm such as the Lesk algorithm. A subjectivity classification and sarcasm detection might also enhance the quality of *SentiStorm*. Another classifier such as Maximum Entropy (MaxEnt) might increase the prediction quality as well. The performance of *SentiStorm* is limited by a bottleneck, which is caused by the high latency components *POS Tagger* and *SVM*. This bottleneck can be improved by the usage of a custom scheduler. A custom scheduler in Storm would allow to schedule a special type of *Bolt* to a specific worker. This would enable a different scheduling strategy of high latency components such as *POS Tagger* and *SVM* compared to other low latency components. Furthermore, it would allow to add more workers or nodes, which exclusively execute high latency components. Another performance enhancement could be achieved by decoupling the topology, which means splitting the topology in a low latency topology and in a high latency topology. The data transport between multiple topologies could be realized by *Apache Kafka*. Kafka is a real-time distributed publish-subscribe messaging system [Gar13, p. 6]. Finally, the high latency components could be implemented in low-level programming languages such as C++ to improve the throughput of *SentiStorm*. The usage of a GPU by *CUDA* or *OpenCL* might also enhance the performance.

Appendix

A.1 Penn Treebank POS Tagset

Table 1: Penn Treebank POS Tagset [Liu12, p. 26]

Tag	Description	Tag	Description
CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential *there*	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
		TO	*to*
JJ	Adjective	UH	Interjection
JJR	Adjective, comparative	VB	Verb, base form
JJS	Adjective, superlative	VBD	Verb, past tense
LS	List item marker	VBG	Verb, gerund or present participle
MD	Modal	VBN	Verb, past participle
NN	Noun, singular or mass	VBP	Verb, non-3rd person singular present
NNS	Noun, plural		
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	proper
POS	Possessive ending	WP$	Possessive wh-pronoun

Bibliography

[ABS14] X. Artola, Z. Beloki and A. Soroa: *A Stream Computing Approach Towards Scalable NLP*, N. C. C. Chair), K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk and S. Piperidis (eds.), *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, European Language Resources Association (ELRA), Reykjavik, Iceland.

[And13] Q. Anderson: *Storm Real-Time Processing Cookbook*, Packt Publishing, 2013.

[AXV+11] A. Agarwal, B. Xie, I. Vovsha, O. Rambow and R. Passonneau: *Sentiment Analysis of Twitter Data*, *Proceedings of the Workshop on Languages in Social Media*, LSM '11, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 30–38, URL `http://dl.acm.org/citation.cfm?id=2021109.2021114`.

[Bab15] P. Babu: *Storm Tweets Sentiment Analysis*, `https://github.com/P7h/StormTweetsSentimentAnalysis`, 2015, (visited February 2015).

[BF10] L. Barbosa and J. Feng: *Robust Sentiment Detection on Twitter from Biased and Noisy Data*, *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 36–44, URL `http://dl.acm.org/citation.cfm?id=1944566.1944571`.

[CL11] C.-C. Chang and C.-J. Lin: *LIBSVM: A Library for Support Vector Machines*, ACM Trans. Intell. Syst. Technol., volume 2(3), (2011), pages

27:1–27:27, URL http://doi.acm.org/10.1145/1961189.1961199.

[CST00] N. Cristianini and J. Shawe-Taylor: *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, 2000.

[CV95] C. Cortes and V. Vapnik: *Support-Vector Networks*, Mach. Learn., volume 20(3), (1995), pages 273–297, URL http://dx.doi.org/10.1023/A:1022627411411.

[DC01] S. Das and M. Chen: *Yahoo! for Amazon: Extracting market sentiment from stock message boards*, *In Asia Pacific Finance Association Annual Conf. (APFA)*.

[DDZC13] I. Dilrukshi, K. De Zoysa and A. Caldera: *Twitter news classification using SVM*, *Computer Science Education (ICCSE), 2013 8th International Conference on*, pages 287–291.

[DRCB13] L. Derczynski, A. Ritter, S. Clark and K. Bontcheva: *Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data*, *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, Association for Computational Linguistics.

[EPGK14] S. Evert, T. Proisl, P. Greiner and B. Kabashi: *SentiKLUE: Updating a Polarity Classifier in 48 Hours*, *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, Association for Computational Linguistics, pages 551–555, URL http://aclweb.org/anthology/S14-2096.

[Gar13] N. Garg: *Apache Kafka*, Packt Publishing, 2013.

[GBH09] A. Go, R. Bhayani and L. Huang: *Twitter Sentiment Classification using Distant Supervision*, Processing, (2009), pages 1–6, URL http://www.stanford.edu/~alecmgo/papers/TwitterDistantSupervision09.pdf.

[GGT13] M. Guerini, L. Gatti and M. Turchi: *Sentiment Analysis: How to Derive Prior Polarities from SentiWordNet*, *Proceedings of the 2013 Conference on*

Empirical Methods in Natural Language Processing, Association for Computational Linguistics, pages 1259–1269, URL http://aclweb.org/anthology/D13-1125.

[Gro15a] A. S. Group: *Apache Storm*, http://storm.apache.org, 2015, (visited February 2015).

[Gro15b] A. S. Group: *Apache Storm*, http://storm.apache.org/documentation/Understanding-the-parallelism-of-a-Storm-topology.html, 2015, (visited February 2015).

[Gro15c] A. S. Group: *Apache Storm*, https://storm.apache.org/documentation/Concepts.html, 2015, (visited February 2015).

[Gro15d] S. A. S. I. Group: *Apache Storm*, http://aclweb.org/aclwiki/index.php?title=SemEval_Portal, 2015, (visited February 2015).

[GSO+11] 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*, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 42–47, URL http://dl.acm.org/citation.cfm?id=2002736.2002747.

[HCL03] C.-W. Hsu, C.-C. Chang and C.-J. Lin: *A Practical Guide to Support Vector Classification*, Technical report, Department of Computer Science, National Taiwan University, 2003, URL http://www.csie.ntu.edu.tw/~cjlin/papers.html.

[HL04] M. Hu and B. Liu: *Mining and Summarizing Customer Reviews*, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, ACM, New York, NY, USA, pages 168–177, URL http://doi.acm.org/10.1145/1014052.1014073.

[Int15] Internetlivestats: *1 Second - Internet Live Stats*, http://www.internetlivestats.com/one-second/#tweets-band, 2015, (visited March 2015).

[JN14] A. Jain and A. Nalya: *Learning Storm*, Packt Publishing, 2014.

[KWM11] E. Kouloumpis, T. Wilson and J. Moore: *Twitter Sentiment Analysis: The Good the Bad and the OMG!*, L. A. Adamic, R. A. Baeza-Yates and S. Counts (eds.), *ICWSM*, The AAAI Press, URL http://dblp.uni-trier.de/db/conf/icwsm/icwsm2011.html#KouloumpisWM11.

[KZM14] S. Kiritchenko, X. Zhu and S. Mohammad: *Sentiment Analysis of Short Informal Texts*, Journal of Artificial Intelligence Research (JAIR), volume 50, (2014), pages 723–762.

[LES12] J. Leibiusky, G. Eisbruch and D. Simonassi: *Getting Started with Storm*, O'Reilly Media, Inc., 2012.

[Liu12] B. Liu: *Sentiment Analysis and Opinion Mining*, Synthesis digital library of engineering and computer science, Morgan & Claypool, 2012.

[Liu15] B. Liu: *Bing Liu Sentiment Lexicon*, http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon, 2015, (visited February 2015).

[LLLY04] B. Liu, X. Li, W. S. Lee and P. S. Yu: *Text Classification by Labeling Words*, *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, AAAI Press, pages 425–430, URL http://dl.acm.org/citation.cfm?id=1597148.1597218.

[MBF+90] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross and K. J. Miller: *Introduction to WordNet: An On-line Lexical Database**, International Journal of Lexicography, volume 3(4), (1990), pages 235–244, URL http://ijl.oxfordjournals.org/content/3/4/235.abstract.

[MKZ13] S. Mohammad, S. Kiritchenko and X. Zhu: *NRC-Canada: Building the State-of-the-Art in Sentiment*

Analysis of Tweets, *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013)*, Atlanta, Georgia, USA.

[MN14] C. Manning and P. Nayak: *Introduction to Information Retrieval*, http://web.stanford.edu/class/cs276/handouts/lecture7-vectorspace-1per.pdf, 2014, (visited February 2015).

[Moh15] S. Mohammad: *Sentiment140*, http://www.saifmohammad.com/WebPages/lexicons.html, 2015, (visited February 2015).

[MPQ15] MPQA: *MPQA Subjectivity Lexicon*, http://mpqa.cs.pitt.edu/lexicons/subj_lexicon/, 2015, (visited February 2015).

[MRS08] C. D. Manning, P. Raghavan and H. Schütze: *Scoring, term weighting, and the vector space model*, *Introduction to Information Retrieval*, Cambridge University Press, 2008, pages 100–123, URL http://dx.doi.org/10.1017/CBO9780511809071.007.

[MVMCPOUnL13] M.-T. Martín-Valdivia, E. Martínez-Cámara, J.-M. Perea-Ortega and L. A. Ureña López: *Sentiment Polarity Detection in Spanish Reviews Combining Supervised and Unsupervised Approaches*, Expert Syst. Appl., volume 40(10), (2013), pages 3934–3942, URL http://dx.doi.org/10.1016/j.eswa.2012.12.084.

[Nie11a] F. Å. Nielsen: *AFINN*, http://www2.imm.dtu.dk/pubdb/p.php?6010, 2011, (visited February 2015).

[Nie11b] F. Å. Nielsen: *A new ANEW: Evaluation of a word list for sentiment analysis in microblogs*, CoRR, volume abs/1103.2903, URL http://arxiv.org/abs/1103.2903.

[NRK+13] P. Nakov, S. Rosenthal, Z. Kozareva, V. Stoyanov, A. Ritter and T. Wilson: *SemEval-2013 Task 2: Sentiment Analysis in Twitter*, *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, Association for Computational

Linguistics, Atlanta, Georgia, USA, pages 312–320, URL http://www.aclweb.org/anthology/S13-2052.

[OOD+13] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider and A. N. Smith: *Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters*, *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, pages 380–390, URL http://aclweb.org/anthology/N13-1039.

[PGEK13] T. Proisl, P. Greiner, S. Evert and B. Kabashi: *KLUE: Simple and robust methods for polarity classification*, *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, Association for Computational Linguistics, Atlanta, Georgia, USA, pages 395–401, URL http://www.aclweb.org/anthology/S13-2065.

[PL08] B. Pang and L. Lee: *Opinion Mining and Sentiment Analysis*, Found. Trends Inf. Retr., volume 2(1-2), (2008), pages 1–135, URL http://dx.doi.org/10.1561/1500000011.

[PLV02] B. Pang, L. Lee and S. Vaithyanathan: *Thumbs Up?: Sentiment Classification Using Machine Learning Techniques*, *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 79–86, URL http://dx.doi.org/10.3115/1118693.1118704.

[PP10] A. Pak and P. Paroubek: *Twitter as a Corpus for Sentiment Analysis and Opinion Mining*, N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner and D. Tapias (eds.), *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, European Language Resources Association (ELRA), Valletta, Malta.

[RGS+14] I. Raina, S. Gujar, P. Shah, A. Desai and B. Bodkhe: *Twitter Sentiment Analysis using Apache Storm*, International Journal of Recent Technology and Engineering, volume 3(5), (2014), pages 23–26.

[RRNS14] S. Rosenthal, A. Ritter, P. Nakov and V. Stoyanov: *SemEval-2014 Task 9: Sentiment Analysis in Twitter*, *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 73–80, URL `http://www.aclweb.org/anthology/S14-2009`.

[San90] B. Santorini: *Part-Of-Speech Tagging Guidelines for the Penn Treebank Project (3rd revision, 2nd printing)*, Technical report, Department of Linguistics, University of Pennsylvania, Philadelphia, PA, USA, 1990.

[SHA12] H. Saif, Y. He and H. Alani: *Semantic Sentiment Analysis of Twitter*, *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, ISWC'12, Springer-Verlag, Berlin, Heidelberg, pages 508–524, URL `http://dx.doi.org/10.1007/978-3-642-35176-1_32`.

[Tec15] H. L. Technoglogy: *SentiWords*, `https://hlt.fbk.eu/technologies/sentiwords`, 2015, (visited February 2015).

[The15] M. Thelwall: *SentiStrength*, `http://sentistrength.wlv.ac.uk`, 2015, (visited February 2015).

[TK08] S. Theodoridis and K. Koutroumbas: *Pattern Recognition, Fourth Edition*, Academic Press, 4th edition, 2008.

[Ton01] R. Tong: *An Operational System for Detecting and Tracking Opinions in On-line Discussions*, *Working Notes of the SIGIR Workshop on Operational Text Classification*, New Orleans, Louisianna, pages 1–6.

[Tur02] P. D. Turney: *Thumbs Up or Thumbs Down?: Semantic Orientation Applied to Unsupervised Classification of Reviews*, *Proceedings of the 40th Annual Meeting on Association for Computational Lin-*

guistics, ACL '02, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 417–424, URL http://dx.doi.org/10.3115/1073083.1073153.

[Twi13] Twitter: *New Tweets per second record, and how!*, https://blog.twitter.com/2013/new-tweets-per-second-record-and-how, 2013, (visited March 2015).

[WFH11] I. H. Witten, E. Frank and M. A. Hall: *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[WWH05] T. Wilson, J. Wiebe and P. Hoffmann: *Recognizing Contextual Polarity in Phrase-level Sentiment Analysis*, *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, Association for Computational Linguistics, Stroudsburg, PA, USA, pages 347–354, URL http://dx.doi.org/10.3115/1220575.1220619.

[ZKM14] X. Zhu, S. Kiritchenko and S. Mohammad: *NRC-Canada-2014: Recent Improvements in the Sentiment Analysis of Tweets*, *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, Association for Computational Linguistics, pages 443–447, URL http://aclweb.org/anthology/S14-2077.