Leopold-Franzens-University Innsbruck

Institute of Computer Science
Databases and Information Systems

Sentiment Detection of Tweets using Factorization Machines and Ensemble Methods

Master Thesis

Benedikt Stricker, BSc

supervised by
Dr. Eva Zangerle

Innsbruck, August 26, 2017

Abstract

Twitter has grown to one of the most popular microblogging websites in recent years and therefore, has become a major source of opinionated texts. Since the length of these Twitter messages is limited to 140 characters, detecting whether such a tweet conveys a positive or negative sentiment is a challenging task and has become a popular field of study in natural language processing.

In this thesis, we propose an approach called *SentTwi* to classify the sentiment of tweets as either positive, neutral or negative. The main goal of *SentTwi* is to demonstrate the suitability of so-called factorization machines and ensemble methods in conjunction with state-of-the-art approaches like part-of-speech tagging, sentiment lexicons and term frequency weighting to detect the sentiment of tweets. A quality evaluation based on the datasets of the International Semantic Evaluation (SemEval) challenge 2016 is performed to evidence that *SentTwi* provides comparable results than other sentiment detection systems.

Contents

Chapter 1

Introduction

Sentiment analysis has become a popular topic in both economy and science. Companies are interested in the public opinion about their products and services. By analyzing product reviews and customer ratings, market researchers are endeavored to find out what people love about their products or what they think about their customer service. Also politicians and political scientists rely on the results of sentiment analysis to investigate the popularity of a political party or the possible outcome of an upcoming election. In science, universities and research groups try to outperform each other by building better and more accurate sentiment analysis tools and classifiers.

At the beginning of sentiment analysis, mostly reviews and documents written by experts have been classified. Nowadays, with the growth of social media and microblogging websites like Facebook and Twitter, millions of people express their opinion and feelings about products, people and their everyday life in short and informal texts. Especially the microblogging platform Twitter has become an interesting target in sentiment analysis. Not only regular citizens use Twitter to write about their life in so-called tweets, also most celebrities and a lot of politicians worldwide are present on Twitter and tweet about their joys and sorrows. This makes Twitter a valuable source to gather information about the opinion and sentiment of a vast amount of people worldwide. Platforms like Social Mention[1] and Social Searcher[2] are using Twitter sentiment analysis to allow interested parties to search and aggregate social media content. Due to the limited length and the informal language used in Twitter messages, it is more difficult to extract sentiment information from tweets than from blogs or reviews, thus conventional sentiment detection approaches have to be rethought and adapted.

[1]`http://www.socialmention.com` - accessed on 19 August, 2017
[2]`https://www.social-searcher.com` - accessed on 19 August, 2017

In traditional sentiment classification applications, supervised learning techniques like naive Bayes, maximum entropy and support vector machines (SVM) are used [Liu12, PLV02]. These approaches are popular as off-the-shelf classifiers because naive Bayes is a simple algorithm and performs quite well in text classification [Lew98] and sentiment classification [PLV02]. The more sophisticated SVM has turned out to be a machine learning all-rounder, applicable to learning problems in various domains which range from image classification [CHV99] to classifying cancerous tissue in bioinformatics [FCD+00] due to their capabilities of handling high dimensional feature spaces well.

Factorization machines (FM) and ensemble methods are two supervised learning concepts used in this thesis. Factorization machines are a relatively new model in machine learning. They are an approach to combine highly accurate factorization models with the generality of feature extraction found in traditional machine learning algorithms like SVMs and linear regression [Ren10]. Ensemble methods are a machine learning technique which combines a set of individually trained classifiers into one integrated prediction [Rok05]. This allows the ensemble to benefit from the knowledge of multiple models instead of relying on the decision of one single classifier.

Both concepts are suited for text classification and recommendation as FMs are able to handle sparse input data well [HDD13, RGFST11], while ensemble methods can improve the performance of a single classifier [SS00, VC14]. However, it is unclear whether FMs and ensemble methods are suitable for sentiment classification of tweets, as no research has explored FMs for this task and only few works have incorporated ensemble methods in sentiment classification of tweets so far [KTM11, MCGVF+13].

To fill this research gap, the goal of this master thesis is to explore the applicability of factorization machines and ensemble methods to classify the sentiment of tweets. Therefore, we present *SentTwi*, a sentiment analysis tool which uses factorization machines and two different ensemble methods to classify whether a given tweet conveys a positive, neutral or negative sentiment. Each tweet is processed in a feature pipeline which transforms the tweet into a numerical representation and extracts features like part-of-speech tags and emoticons. We use the datasets provided by the Semantic Evaluation challenge (SemEval) to be able to compare the evaluation results of *SentTwi* with other participants of this international competition. The evaluation of *SentTwi* shows that FMs and the employed ensemble methods are suitable for sentiment classification of tweets and achieve similar performance results as other classification techniques like SVMs.

This thesis is structured as follows. In Chapter 2, we give some background information regarding the microblogging service Twitter, provide a brief overview about sentiment classification and describe SemEval, a workshop on semantic evaluation. This chapter explains also the novel classification techniques factorization machines and ensemble methods. Chapter 3 presents work related to this thesis. The approach and implementation of *SentTwi* is presented in Chapter 4. Next, Chapter 5 introduces the metrics used to evaluate *SentTwi* and provides the obtained results, while Chapter 6 discusses these findings. Finally, Chapter 7 concludes the thesis and presents future work.

Chapter 2

Background

To comprehend the main aspects of this thesis, the following chapter imparts some basic knowledge. Since this thesis is about classifying tweets, an introduction to Twitter is given first. Afterwards, the main principles of sentiment classification including supervised and unsupervised learning as well as some typically used components in sentiment detection are presented. The next section of this chapter is about the Semantic Evaluation tasks which are used to compare our findings and scores with other classifiers. The mathematical definition and working principles of factorization machines and an overview of ensemble methods, together with two popular families of ensemble methods are presented in the last two sections of this chapter.

2.1 Twitter

Twitter[1] is a microblogging service that launched in July 2006. Today, more than ten years later and after a rapid growth, Twitter counts about 328 million monthly active users with more than 80% experiencing Twitter on mobile devices[2]. The success of Twitter compared to other microblogging websites and social media platforms can be summed up in three key aspects:

Brevity

Each Twitter user can send a 140-characters limited message (*tweet*) which is visible to all other users. Instead of writing a lengthy, several paragraphs long blog, users can put down their feelings, opinions and thoughts in a few short and concise words.
By *retweeting*, a user can forward someone else's tweet and share this

[1]`https://twitter.com` - accessed on 19 August, 2017
[2]`https://investor.twitterinc.com/results.cfm` - accessed on 19 August, 2017

tweet to his/her own followers to spread the message. Besides forwarding, users can also reply to a specific tweet to create a conversation between two or more users. Replying is realized with *mentions*. Mentions are tweets that contain another username preceded by the *@-sign* (e.g., *@twitter*). Tweets can contain one or more mentions anywhere in the tweet. If a user is mentioned by someone, he/she gets notified and can respond. If a tweet begins with *@username*, the tweet is a reply to this user and will be additionally shown in the home timeline of the two participants.

Hashtags

By preceding a word with the *#-sign*, users can annotate and group their tweets into different topics. This prefixed word is called *hashtag* and allows users to search and find tweets about a certain topic. If enough users tweet the same hashtag within a small amount of time, this hashtag becomes a *Trending Topic* and will be promoted to more users to participate in this trend.

Followers

A user can subscribe to tweets from other users (*following*). When following someone, each tweet will appear in the follower's own Twitter timeline. Following on Twitter is unidirectional, which means that users do not have to follow their followers too. This allows users to get updates on tweets from friends, celebrities and communities.

2.2 Sentiment Analysis

Every subjective text contains to some extent sentiment. This sentiment expresses the writer's emotions, opinions or feelings. *Sentiment analysis* (also known as *opinion mining*) focuses on extracting and classifying sentiment in all kinds of evaluative texts [PL08]. With the rise of Web 2.0, blogs, forums and social media, people have generated more opinionated data than ever before. Since then, sentiment analysis has emerged to one of the most active topics in natural language processing [Liu12] with different real-life applications. One popular use is the classification and summarization of product reviews: Instead of reading through hundreds of reviews, potential customers would like to see at a glance which reviews are helpful and what other customers think about the product's features. Hu and Liu [HL04] proposed a system for mining and summarizing product reviews by finding product features which are mentioned by customers in an opinion-related context (e.g., "The pictures of this camera are absolutely amazing."). Depending on the classification goal, sentiment can either be classified in a positive and a negative class or a three-class based rating which uses in addition to the two mentioned classes a third neutral or objective class to denote documents that do not contain any notion of sentiment [Liu12].

2.2.1 Types of Learning Methods

As a text classification problem, supervised and unsupervised learning methods can be applied to predict the sentiment class of a given document. Although most techniques for sentiment classification are based on supervised methods [Liu12], we will briefly present both learning methods in the following.

Supervised learning methods

Supervised learning methods use training data together with the corresponding output (*labels* or *target values*) and return a classification function (*classifier*). The training data is either annotated by a supervisor (human expert) who labels the data according to the used classification (e.g., binary or multi-class), or automatically annotated based on measurements or opinionated data. After training, the classifier can predict labels of any unseen data based on the training data.

Unsupervised learning methods

In *unsupervised learning*, the learning method receives neither labels, nor any feedback and therefore tries to find patterns in the given data to predict future input data. The most common use cases for unsupervised learning methods are finding clusters of similar data (*clustering*) and *dimensionality reduction*, which finds a smaller set of features while retaining the expressiveness of the original data.

In sentiment classification, we want to compute the sentiment of a document according to a fixed set of classes, therefore primarily supervised learning methods are used. Techniques like naive Bayes, maximum entropy and support vector machines have been proven to be accurate for binary classification problems in various domains [PLV02, DLP03, CNM06]. Altough, Turney [Tur02] presented a simple unsupervised method to classify reviews based on the semantic orientation of adjectives and adverbs. Another unsupervised approach is the use of sentiment lexicons to lookup the sentiment of a single word or a phrase to calculate the overall sentiment of a document [DLY08].

	Tweet
T1	This Pizza is better than this Burger
T2	This Burger is not bigger than this Pizza

Table 2.1: Example Tweet T1 and T2

Term	better	bigger	burger	is	not	pizza	than	this
T1	1	0	1	1	0	1	1	2
T2	0	1	1	1	1	1	1	2

Table 2.2: Feature Vectors of T1 and T2 Using Term Frequency

2.2.2 Components of Sentiment Analysis Systems

To compute the sentiment of a document, there exist some established methods and components which will be explained in the following paragraphs.

Feature Extraction

In order to allow a classifier to analyze the sentiment of the given text documents, we need to transform their textual representation into a numerical form which the classifier algorithm understands. This numerical representation is called *feature vector*, or short *features*, and is stored in a two dimensional matrix where each row denotes one tweet and the extracted features are described by the column index. In the following pages, the two example tweets seen in Table 2.1 are used to exemplify frequently used features [PL08] in sentiment analysis.

Term based features In text classification, a simple way of representing a single document as a feature vector is by counting the frequency of each term in the document. For every document in the corpus, the occurrence of each term is counted and the corresponding entry in the feature vector is set to its frequency. This is referred to as *term frequency* $tf_{t,d}$ of term t in document d.
Table 2.2 shows the calculated features for the two example tweets of Table 2.1. It is easy to see that the words “burger”, “is”, “pizza” and “than” are present in both tweets once, so each tweet has a tf-value of 1 for the given term. Likewise has the term “this” for both tweets a value of 2 as it occurs twice in both tweets. Generally spoken, each row consists of the frequency values for each term contained in each tweet.
Pang et al. [PLV02] used a simpler feature calculation by using *presence* rather than *frequency* and obtained better performance in classifying the

Term	better	bigger	burger	is	not	pizza	than	this
df	1	1	2	2	1	2	2	2
idf	0.3	0.3	0	0	0.3	0	0	0

Table 2.3: Calculated Document Frequency *df* and Inverse Document Frequency *idf* of the Example Corpus

sentiment of movie reviews. Presence can easily be realized by binary values indicating whether a term is present in the document (value 1) or not (value 0).

Instead of using single words (*unigrams*) as terms, it can be also useful to consider higher-order n-grams like *bigrams* ($n = 2$) or *trigrams* ($n = 3$). The bigrams for the example tweet T1 are "This Pizza", "Pizza is", "is better", "better than", "than this" and "this Burger". The trigrams for the same tweet are "This Pizza is", "Pizza is better", "is better than", "better than this" and "than this Burger". Which n-grams provide better results cannot be said a priori. Sometimes bigrams and trigrams work better for product reviews like in the findings of Dave et al. [DLP03], whereas Pang et al. [PLV02] achieved higher accuracy when preferring unigrams over bigrams in the movie review setting.

The *tf-idf* model [MRS08] is a similar but more advanced metric which tries to find the discriminative terms which account most for the assigned sentiment of a document, by measuring the relevance of a term related to the corpus. Instead of using just the term frequency tf, a second criteria *idf* is added which boosts the score of terms which occur rarely in the document collection and reduces it for very frequent terms. To achieve this behavior, we make use of the *document frequency* df_t which is simply defined as the number of documents that contain the term t. By computing the logarithm of the document frequency relative to the number of documents N, we obtain the *inverse document frequency idf*. More formally, $idf_t = log\frac{N}{df_t}$. Therefore, the inverse document frequency of the term "burger" is 0 because $log\frac{2}{2} = 0$. The logarithm is only used for dampening the effect of the inverse frequency, so it is not important which logarithm is taken, as long as the same logarithm is used throughout the feature calculation.

Table 2.3 shows the document frequency *df* and the inverse document frequency *idf* for the corpus consisting of the all terms appearing in the two example tweets T1 and T2 introduced in Table 2.1. Finally, the tf-idf score as the discriminative power of a word regarding the whole corpus is calculated as $tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$. In Table 2.4 it can be seen that only infrequent terms (with non-zero idf) have a tf-idf score greater than zero.

Term	better	bigger	burger	is	not	pizza	than	this
T1	0.3	0	0	0	0	0	0	0
T2	0	0.3	0	0	0.3	0	0	0

Table 2.4: Computed tf-idf Weighting for Example Tweet T1 and T2

To summarize, the tf-idf weight increases if a term occurs often in a document while appearing only in a small number of documents and decreases if a term is very frequent across the corpus. In literature, there exist different versions of the tf-idf statistic which differ mostly in the calculation of the tf and idf factors [MRS08]. Although tf-idf is widely used in topic classification, Paltoglou et al. [PT10] have shown that some tf-idf variants can improve the classification accuracy in sentiment analysis compared to use of term frequency and term presence.

Part-of-Speech tags Part-of-Speech (POS) tags are another kind of features which are often used in semantic analysis [PL08]. A POS tagger assigns each term in a sentence its grammatical word class (e.g., noun, verb, adjective). This allows to focus on parts of speech like adjectives and adverbs which can be a good indicator of sentiment [PL08]. POS tags can also be added to the term frequency feature set to keep track of the frequency of each part of speech. Section 4.3 will give more details about the POS tagger used in this thesis, as well as the exact use of the calculated POS tag features in the feature extraction process.

Negation Negation words, like "no" and "never", are crucial as they can change the sentiment from positive to negative or vice versa. Handling negation is a nontrivial task as negation can also be expressed in a more subtle way with irony or sarcasm. A popular and simple method to incorporate negation is by adding artificial words [WBR$^+$10]: i.e., attaching "NOT" to a word following a negation word, so "This phone is not perfect" is changed to "This phone is not perfect_NOT". Handling complex kinds of negation, Wiegand et al. [WBR$^+$10] discusses advanced modeling techniques like using heuristic rules and supervised machine learning.

Classification

In the classification step, we use the feature matrix obtained in the feature extraction phase together with the ground truth labels as input for our classifier. The ground truth is a single vector containing the sentiment of every document. Figure 2.1 depicts the process of this procedure. After training the classifier with the training data and the truth

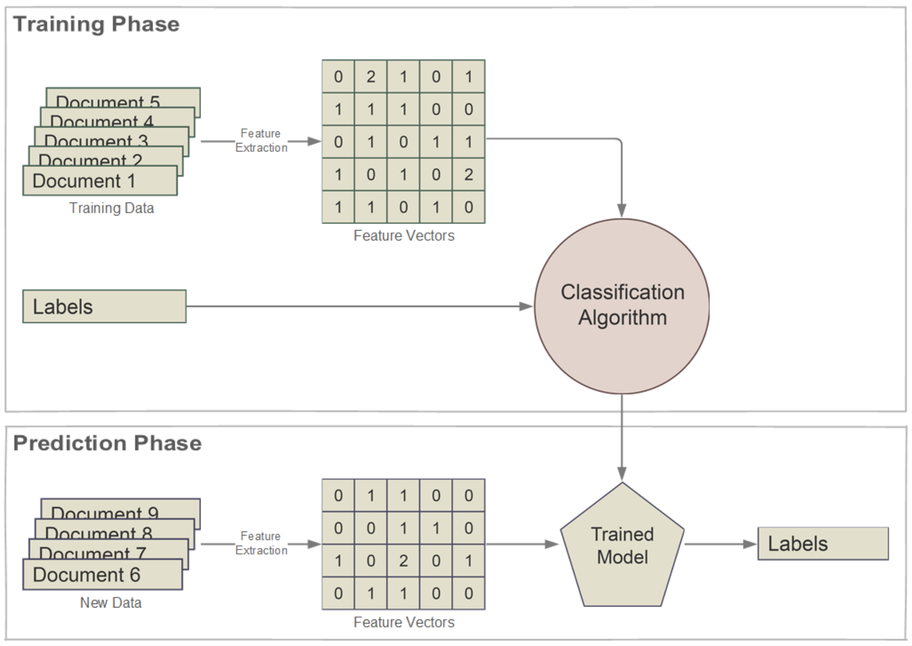

Figure 2.1: Supervised Learning

labels, we obtain a model which can be used to predict new documents. For this unseen data, we need to extract their features using the same feature extraction approach as for the training data. The concepts of the classifiers used in this thesis are presented in more detail in Section 2.4 and 2.5. Their implementation in *SentTwi* is given in Section 4.5.

2.2.3 Challenges in Sentiment Analysis

In contrast to topic-based classification, which focuses on classifying documents based to their subject (e.g., science, business, sports), sentiment-based classification has to face some new challenges due to the high subjectivity of the documents. Users can express their sentiment in a subtle manner or use sarcasm to say the opposite of what they mean. The tweet “Fleetwood Mac is the only thing getting me through this Monday morning” expresses a positive sentiment in a very subtle manner without using any obviously positive words. The same goes for sarcastic tweets like “I can’t express how much I love shopping on Black Friday.”, which reads like a positive tweet but should convey a negative attitude.

2.3 SemEval

SemEval[3] (Semantic Evaluation) is an international workshop which has evolved from the SensEval series. Since 2012, this workshop is held on an annual basis and focuses on the exploration of the capabilities of semantic analysis systems. Each year, the organizers assign various tasks for teams from around the world to compete against in challenges like automatically ranking relevant answers in Community Question Answering Forums[4] or parsing semantic dependencies in Chinese sentences[5]. A very popular task in the last years is the semantic evaluation of tweets which attracted 43 teams in 2016 [NRR+16] and is officially referred as *SemEval-2016 Task 4: Sentiment Analysis in Twitter*[6]. Due to the popularity of this topic, five independent subtasks are provided. *Subtask A: Message Polarity Classification* is the type of sentiment analysis described in this thesis and forms the basis to compare our system with other teams. Two of the other four subtasks are about tweet quantification (estimating *how many* tweets have a positive or negative view of a given topic) and two about tweet classification on a given topic (estimating the tweet's sentiment towards a *given topic* instead estimating the overall sentiment). The different datasets for subtask A, created and provided by the task organizers are shown in Table 2.5. The complete training set for 2016 consists of all released datasets from 2013 to 2015 and the training and development sets of 2016. The test set is the official test data used for the evaluation of SemEval 2016 participants. Since some tweets appear in multiple datasets, the number of total and unique tweets in the composed datasets differs slightly. The number of tweets is also lower compared to the dataset statistics presented by the organizers [NRR+16] as some tweets can not be accessed anymore.

2.4 Factorization Machines

Factorization machines (FM) are a general framework for factorization models. FMs have been proposed by Rendle [Ren10] and can be seen as a combination of support vector machines and factorization models. Like SVMs, FMs are general predictors which can be trained with feature vectors. By using factorized parameters, FMs achieve the high-prediction accuracy known from factorization models like matrix and tensor factorization [Ren12a]. Similar to SVMs with a polynomial kernel, FMs can model all interactions between features up to a given degree d between n input variables in the feature vector $\mathbf{x}$. A two-way FM ($d = 2$) models all

[3]`http://alt.qcri.org/semeval2016/` - accessed on 19 August, 2017
[4]`http://alt.qcri.org/semeval2016/task3/` - accessed on 19 August, 2017
[5]`http://alt.qcri.org/semeval2016/task9/` - accessed on 19 August, 2017
[6]`http://alt.qcri.org/semeval2016/task4/` - accessed on 19 August, 2017

Dataset	**positive**	**neutral**	**negative**	**total (unique)**
2013	4,225	5,283	1,678	**11,186**
2014	764	549	147	**1,460**
2015	915	985	326	**2,226**
2016 Train	2,533	1,663	690	**4,886**
2016 Dev	704	632	325	**1,661**
2016 DevTest	815	584	265	**1,655**
Total Train	**9,956**	**9,696**	**3,422**	**23,074 (22,700)**
Test	**5,788**	**8,458**	**2,512**	**16,758**
Total	**15,744**	**18,154**	**5,934**	**39,832 (39,395)**

Table 2.5: SemEval Data Statistics

single and pairwise interactions between the feature variables. This allows the FM to capture similarities even in very sparse feature vectors $\mathbf{x}$.

The model equation of a two-way FM is defined as:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{2.1}$$

where w_0, $\mathbf{w}$ and $\mathbf{V}$ are the three model parameters (collectively referred as Θ) the FM has to estimate:

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{V} \in \mathbb{R}^{n \times k} \tag{2.2}$$

The dot product of two vectors of size k is denoted by $\langle \cdot, \cdot \rangle$, where $k \in \mathbb{N}_0^+$ is a hyperparameter that defines the dimensionality of the factorization:

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \tag{2.3}$$

The first term of the model equation, that is w_0, is the global bias to offset all model predictions. The second part represents the unary interactions of each input variable x_j where w_i models the weight of the i-th variable. The third part with the two nested sums contains the pairwise interactions of the two input variables x_i and x_j. The dot product $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ models these interactions by factorizing them. This is the important key point of FMs and the reason why they can handle highly sparse data well. Instead of modeling the pairwise interaction with an independent parameter $w_{i,j}$ like SVMs with a quadratic kernel, FMs model pairwise interaction with a factorized parameter $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$.

This characteristic holds for higher order FMs ($d > 2$) and SVMs with a polynomial kernel of arbitrary degree.
Although the two nested sums in Equation 2.4 suppose a computational complexity of $O(k\,n^2)$, Rendle [Ren10] shows a proof for linear complexity: Due to the factorization of the pairwise interactions $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$, no model parameter depends directly on two variables. This allows to reformulate the model equation to:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2 \right) \tag{2.4}$$

which has a complexity of $O(k\,n)$ - i.e., has linear complexity.

2.4.1 Learning Methods

To learn the model parameters Θ (w_0, $\mathbf{w}$ and $\mathbf{V}$), the three learning methods Stochastic Gradient Descent (SGD), Alternating Least Square (ALS) and Markov Chain Monte Carlo Inference (MCMC) have been proposed by Rendle [Ren12a]:

Stochastic Gradient Descent Stochastic gradient descent tries to find the optimal model parameters by iterations and incremental adaptions of the model parameters. SGD is a popular optimizing method as it has low computational complexity and low storage complexity. One critical part when using factorization machines with one of the three learning methods is the right choice of the following hyperparameters:

- **Learning Rate** — The learning rate parameter is used to control the step size of the iteration. If the step size is too high, SGD will not converge as it will miss or fluctuate around the optimum. If it is too low, the optimization process will take too long as too many potentially unnecessary computations are performed before the optimum is found.

- **Regularization** — Regularization parameters are used to reduce overfitting. FMs using SGD or ALS allow to specify regularization parameters for either all model parameters or three separate parameters for the global bias w_0, one-way interactions $\mathbf{w}$ and pairwise interactions $\mathbf{V}$. Finding the correct regularization values is usually done using a grid search, which is very time-consuming and thus, a well defined search space is essential.

- **Initialization** — The model parameter $\mathbf{V}$ has to be initialized with non-constant values to work correctly with all of the three learning methods. The initialization parameter is used as the standard deviation for a normal distribution which samples the start values for $\mathbf{V}$.

With *SGD with Adaptive Regularization (SDGA)*, Rendle [Ren12b] presented a method to automatically adapt the regularization parameters during the training of the model. This is done by using a separate validation set which optimizes the regularization values during each iteration. This approach makes learning the model parameters easier and faster as no regularization parameters have to be specified and therefore, no expensive grid search is needed.

Alternating Least Square Alternating least squares or coordinate descent is a learning method which finds each of the three model parameters in Θ independently. The optimal value for each model parameter is found while fixing the remaining parameters. This is repeated until the joint optimum of all model parameters is found.

The main advantage of ALS over SGD is that this method only needs to find the two hyperparameters **regularization** and **initialization**. This makes ALS easier to use while performing as good as SGD.

Markov Chain Monte Carlo Inference The MCMC learning method uses Bayesian inference technique and Gibbs sampling to integrate the regularization parameters into the model. These regularization parameters are integrated by sampling from their distribution and iteratively updating the parameters until the optimum values for Θ are identified.

By automatically determining the regularization parameters, the only hyperparameter for MCMC that has to be specified is the **initialization**. Experiments conducted by Rendle [Ren12a] have shown that the MCMC learning method slightly outperforms the other learning methods in predicting movie ratings and recommending movies. This makes MCMC inference the learning method of choice to find the model parameters for a FM, as it is also the fastest (no extensive grid search) and easiest (single hyperparameter) method to use.

2.5 Ensemble Methods

Ensemble methods are a technique which combines multiple classifiers to achieve better prediction results compared to a single learning method. An ensemble consists of classifiers whose output is integrated into a single prediction. Note that most statements about ensemble methods

hold for both learning tasks, classification and regression[7]. Therefore, within the scope of this section, we will use the term *classifier* to denote a learning method that is applicable to both tasks.
It is important to know that not every combination of classifiers automatically achieves a better performance than the individual classifiers. A set consisting of identical classifiers will achieve no improvement as each classifier produces the same output, irrespective of the combination of these outputs. Hansen and Salamon [HS90] determined two conditions an ensemble has to fulfill in order to gain performance. First, each classifier must have an error rate less than $\frac{1}{n}$, with n representing the number of possible output classes. In other words, the classifier has to perform better than simply guessing the correct output values on new data. The second condition is satisfied if the classifiers are diverse. Diverse classifiers have to be independent and produce different errors on the same unseen data. An ideal ensemble would consist of classifiers that generalize well and disagree as much as possible [KV94].
Rokach [Rok05] describes four characteristics to classify various ensemble methods, which will be described in detail in the following paragraphs.

- **Inter-classifier Relationship** — How does each classifier affect the other classifiers in the ensemble?
- **Combining Method** — How does the ensemble combine the individual predictions to one final output?
- **Ensemble Diversity** — How does the ensemble obtain diversity between the learners to become efficient?
- **Ensemble Size** — How should the size of the ensemble be selected to achieve better performance?

Inter-classifier Relationship

Classifiers can either have an effect on other classifiers (*sequential type*) or act independently (*concurrent type*).
Ensemble methods using a sequential approach build their ensemble iteratively. In each iteration, a new classifier is trained based on the knowledge of the previous classifiers. Figure 2.2 depicts the process of training an ensemble with a sequential approach. Each classifier h_i is trained after the previous classifier h_{i-1}. The input data D_i is a distribution based on the original data D and some knowledge feedback

[7]Regression predicts continuous data (e.g., stock prices or temperature) rather than outputting categorical values (e.g., positive/negative sentiment or movie ratings) like in classification problems.

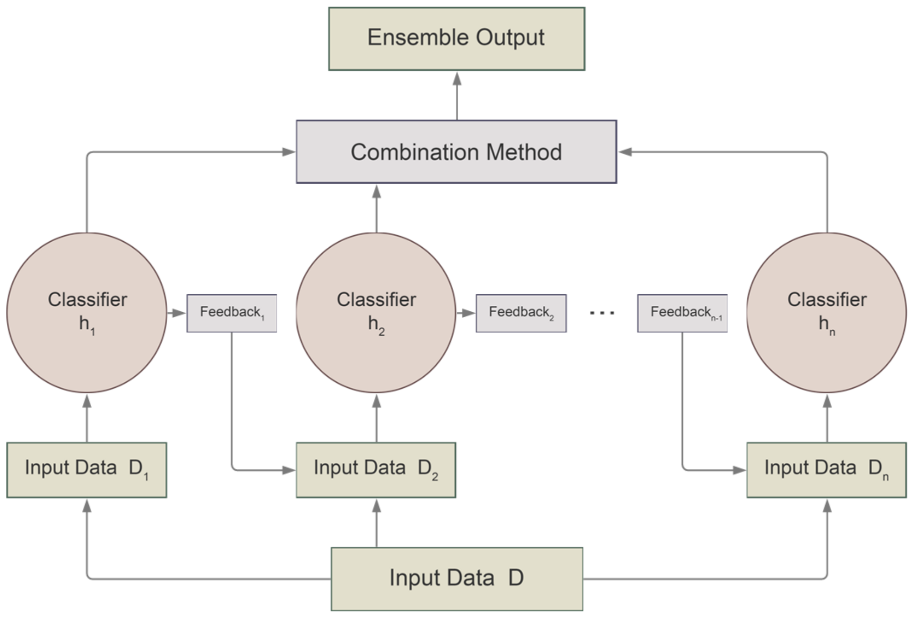

Figure 2.2: Ensemble Using a Sequential Approach

(e.g., performance, error, weights) of h_{i-1}. Each individual prediction is then combined in a final ensemble output. The advantage of combining the output of all classifiers into a final value instead of using only the output of the last classier (i.e., classifier with the most knowledge) is that the earlier classifier have a balanced view over the whole training data, whereas subsequent classifiers can focus more on the mistakes of their predecessors. An example of sequential methods is the family of boosting algorithms described in Section 2.5.1.
Concurrent methods train all classifiers at once. To avoid creating a set of identical classifiers, this approach focuses on the partition of the whole dataset into multiple diverse datasets. All classifiers are then trained concurrently on the given input data. Figure 2.3 visualizes this methodology. Different datasets D_i are derived from the original input dataset D. Each classifier h_i is then trained on the given dataset. Finally, the ensemble combines the output values of the individual classifiers with a combination method to a final prediction output. Bagging is a popular concurrent ensemble method and is described in detail in Section 2.5.2.

Combining Method

As each classifier in the ensemble outputs some predictions, a single and final value for the ensemble as a whole has to be determined. Several

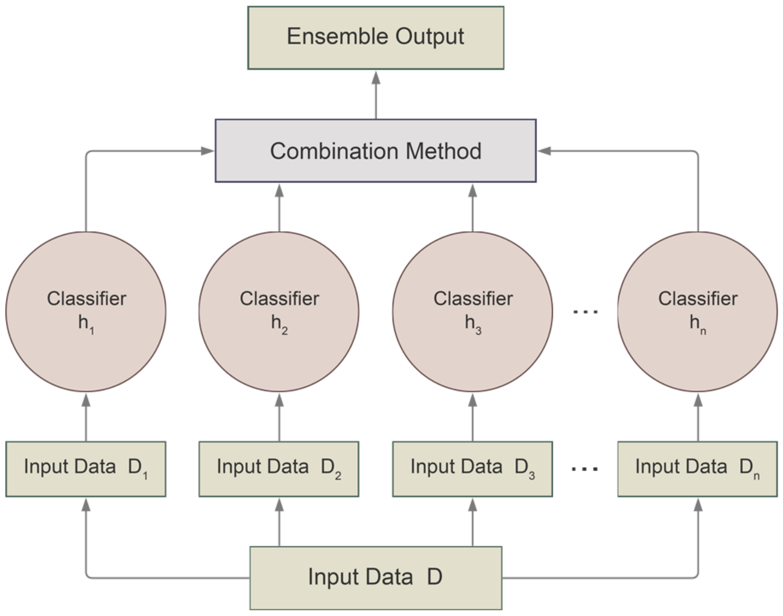

Figure 2.3: Ensemble Using a Concurrent Approach

methods exist to combine the set of predictions, which vary from simple averaging to more complex methods like the application of separate classifiers for meta-combining.

Simple Combining Methods The most popular simple combining methods are *averaging* and *voting*. Simple combination can be described as a method to find the combined output value h_{final} for a given set of N learners $\{h_1, ..., h_N\}$ where the predicted value of instance h_i for input $\mathbf{x}$ is denoted as $h_i(\mathbf{x})$. Depending on the machine learning task, $h_i(\mathbf{x}) \in \mathbb{R}$ for regression and $h_i(\mathbf{x}) \in C$ for classification, where C is the finite set of possible class labels $\{c_1, ...\ c_m\}$.

Averaging Averaging computes the combined output in numerical applications, where the output $h_i(\mathbf{x})$ for the learning model h_i is a numerical value that can be averaged, e.g., $h_i(\mathbf{x}) \in \mathbb{Z}$ or $\mathbb{R}$. To average the outputs of the learning models, each output can be equally weighted (*simple averaging*) or outputs of some learners can be given more importance (*weighted averaging*). The final output h_{final} of a simple averaging ensemble is calculated as follows:

$$h_{final} = \frac{1}{N} \sum_{i=1}^{N} h_i(\mathbf{x}) \tag{2.5}$$

Weighted averaging methods assign each learning model h_i a weight w_i to regulate their impact on the combined output. To simplify computation and comparability with other methods, weights are usually positive ($w_i \geq 0$) and sum up to 1 ($\sum_{i=1}^{N} w_i = 1$). Weighted averaging can mathematically be written as:

$$h_{final} = \sum_{i=1}^{N} w_i \, h_i(\mathbf{x}) \tag{2.6}$$

Voting Voting is a combination method that is used to combine outputs of an ensemble used for classification or any other task that produces categorical outputs.
Majority voting is a simple voting schema that selects the class that has been predicted by more than the half of classifiers. If no class received more than half of the votes, the ensemble rejects the input and does not predict anything. The disadvantage of this combination method is that the possibility of a rejection is higher if the classification task has a lot of possible classes.
Plurality voting (or uniform voting) eliminates the possibility of a rejection as it predicts the class with the highest number of votes. Mathematically, it can be written as:

$$h_{final} = max_{c_j \in C} \sum_{i=1}^{N} h_i^{c_j}(\mathbf{x}) \tag{2.7}$$

where $h_i^{c_j} \in \{0, 1\}$ evaluates to one if classifier h_i predicts class c_j and zero otherwise.
Analogous to the weighted average method, there exist a *weighted voting* schema. This combination method is often used if some of the individual classifiers are known to perform better than others and thus their vote should receive more weight. Ideally, the weights w_i should have the same constraints ($w_i \geq 0$ and $\sum_{i=1}^{N} w_i = 1$) as in the weighed average method. The weighted voting schema is calculated as follows:

$$h_{final} = max_{c_j \in C} \sum_{i=1}^{N} w_i \, h_i^{c_j}(\mathbf{x}) \tag{2.8}$$

Meta-learner Combining Methods Meta-combining methods use an additional meta-learner to combine the outputs of the learners. Although there exist a number of different meta-combining methods [CS93, CS97, SF01, LL08], we will present *stacked generalization* [Wol92] as an example of these methods.
Stacked generalization (or stacking) is a general framework developed by Wolpert [Wol92]. In stacking, a meta-learner receives the predicted

output values of the base-level learning models as training data to predict the final output. The original dataset is divided into two subsets. The first subset is used to train the base-level learners, while the second subset forms the training data for the meta-learner. After the first tier of learners is trained using the first subset, they predict the labels of the second subset. These predicted values form the training set for the meta-learner. The meta-learner is then trained on this training set together with the original labels of the subset. This allows the meta-learner to use the disagreement of the base-level classifiers to correct their improper training by learning their behavior from their individual prediction values.

Ensemble Diversity

Using diverse classifiers are a very important factor when constructing an ensemble. An ensemble consisting of similar learning algorithms trained with the same dataset will most likely degenerate to one single classifier, which would be no improvement at all. Rokach [Rok05] states two general methods to generate diversity: either by *manipulating the classifier* or by *manipulating the training data*. Dietterich [Die00a] mentions a third technique for constructing ensembles: *manipulating the output targets*.

Manipulating the Classifier A simple way to change the behavior of each classifier instance is by injecting randomness into the learning algorithm. Dietterich [Die00b] proposed a decision tree algorithm that does not choose the best attribute at each node in the tree like standard decision trees, but randomly selects one of the 20 best suited attributes. The author showed that an ensemble of 200 classifiers using this decision tree variant performs statistically significantly better than a single decision tree.
Neural networks can also be used as a base algorithm for diverse ensembles by training each network with a randomly chosen set of initial weights [Pol90].

Manipulating the Training Data One way to manipulate *training examples* is by training each classifier on a different subset of the whole training data. This method is effective for an ensemble with unstable learning algorithms (i.e., minor changes in the training data cause major changes in the predictions). Examples of unstable learning algorithms are decision trees and neural networks, whereas SVMs, linear regression and k-nearest neighbor are stable algorithms.
Instead of changing the set of training data, it is also possible to change the set of *input features* each classifier is trained on. Each individual

classifier is then trained on all training examples but only with a subset of features. This method can be useful if the data consists of a high number of redundant features. However, the first method is often preferred over this method as it is easier to find more training examples that can be split into subsets than finding new input features that describe the data.

Manipulating the Output Targets This technique changes the target values that are given to each individual classifier. Dietterich and Bakiri [DB95] proposed a method called *error-correcting output coding* for multiclass learning algorithms by encoding each class as a binary string of fixed length. For each classifier in the ensemble, the set of possible classes is randomly separated into two subsets. The target values of the training data are then relabeled such that all classes belonging to the first subset are mapped to the derived class 0 while the original classes in the second subset get the label 1. Each classifier is then trained on the training data with different labels. New data can then be classified by letting each classifier predict the derived output label 0 or 1 meaning that the data belongs either to a original class in the first or second subset. Each class in the corresponding subset receives a vote and the final prediction is then determined by the class with the highest number of votes.
In other words, each class is described as a binary code word and each classifier is trained to predict a given bit of this code word. Predicting new data is then done by combining the binary output of the individual classifiers into a new code word and choosing the class whose code word has the smallest Hamming distance to the predicted code word.

Ensemble Size

The last factor for building successful ensembles is to choose the correct number of classifiers. A bigger ensemble generally improves the overall accuracy (to some point) but also increases the computational complexity as more classifiers have to be coordinated. As each ensemble responds differently to changes in the number of classifier, the ensemble size has to be found manually for each problem.

2.5.1 Boosting

Boosting [Sch90] is the concept of a learning method that sequentially improves an arbitrary learning algorithm. Boosting trains a series of classifiers on different training data in so-called *boosting rounds*. In each round, a new classifier is trained on a training set composed of training samples based on the performance of the previous classifiers. The final prediction is then made by combining the output of all classifiers.

	Training Data
D (original)	1, 2, 3, 4, 5, 6
D_1	5, 2, 4, 3, 6, 1
D_2	4, 6, 5, 3, 1, 4
D_3	4, 2, 3, 4, 4, 5
D_4	4, 4, 6, 5, 4, 4

Table 2.6: Boosting: Example Training Sets for AdaBoost with Four Rounds of Boosting. Assuming Training Example 4 is Hard to Classify and Example 1 is an Easy One. Therefore, AdaBoost Samples the Hard Examples More Often Since These are More Prone to Misclassifications.

Freund and Schapire [FS96] developed a concrete algorithm called AdaBoost based on this Boosting concept. AdaBoost chooses the training data for subsequent classifiers based on the misclassifications of the classifier in the previous boosting round. Including misclassified instances into the training progress is achieved by *weighting* each instance. The weight of the instance is increased if the example has been misclassified, while the weight decreases with correct classifications. Therefore, it is required that the base algorithm supports weighted training instances. If this is not the case, *resampling* has to be used. Resampling is done prior to the training process and samples the training data according to the assigned training weights. The higher the weight is, the higher the probability that this instance is chosen for the new dataset. This results in multiple instances of "difficult" examples and few occurrences of "easy" instances.

In the first boosting round, each instance x_i in the original dataset D has the same weight and thus, the same probability that it will be selected as a training example in the training set D_1 for the first classifier h_1. After evaluating the performance of h_1, the weight for instance x_i is decreased if h_1 classifies x_i correctly else the weight will be increased. This forces the next classifiers to concentrate on "hard" examples which get often misclassified instead of dealing with "easy" examples that always get classified correctly. Table 2.6 shows an example how AdaBoost with four boosting rounds resamples each training set D_i for h_i. Assuming that example 4 is a "hard" example and example 1 an "easy" instance, it can be seen that the latter classifiers have multiple instances of example 4 and therefore will focus more on the correct classification of this example.

The final prediction is determined by a weighted vote by giving greater weights to classifiers with lower error (i.e., fewer misclassifications).

Experiments comparing AdaBoost with other ensemble methods and learning algorithms demonstrated that AdaBoost often performs very

	Training Data
D (original)	1, 2, 3, 4, 5, 6
D_1 (resample)	3, 4, 1, 4, 5, 3
D_2 (resample)	6, 5, 2, 3, 6, 1
D_3 (resample)	5, 4, 5, 5, 1, 3

Table 2.7: Bagging: Example Training Sets for an Ensemble of Three Classifier. Each Resampled Dataset D_i is a Random Subset of the Original Dataset D.

well but is sensitive to noise [OM99, Die00a]. This is because the weight of noisy examples will increase drastically as they are often misclassified. However, AdaBoost can greatly outperform other methods like bagging or randomized trees on appropriate data (e.g., less noise or enough data).

2.5.2 Bagging

Bagging (B*ootstrap* AGG*regat*ING) [Bre96] is one of the most well-known concurrent methods to generate an ensemble of learners. Bagging is a simple bootstrap algorithm that creates a set of classifiers and trains each classifier h_i concurrently on a random distribution of the original training data D. The new training set D_i for each h_i is created by sampling from D with replacement until D_i has the same amount of examples as the original dataset. As sampling with replacement is used, some training instances may be repeated and others will not be present in the sampled dataset.
Table 2.7 shows an example run of bagging with three classifiers, the original dataset D and three resampled training datasets, each consisting of six instances. For example, the training set D_1 contains examples 3 and 4 twice while 2 and 6 are missing. This makes the classifier trained on this data an “expert” for data similar to the training examples 3 or 4 whereas the error for examples like 2 and 6 probably will be high, as the classifier has never seen such data.
Bagging predicts new data instance as follows: Each classifier calculates an individual output which is then aggregated by the ensemble in a final prediction value. Depending on the task, bagging uses voting for classification and averaging for regression. The combination of the output of a set of diverse “experts” compensates the higher error-rates of the individual learners. Since the classifiers are trained concurrently, bagging features two benefits compared to sequential methods like Boosting. First, the whole training process of the ensemble can be computed in parallel and thus bagging is very attractive for computing on multi-core and parallel computers which can reduce the training time drastically.

Second, the random drawing makes the sampled training sets highly independent which is a important factor for a successful ensemble. Different work [OM99, Die00a] shows that bagging is appropriate for most problems as it is insensitive to noisy data and usually performs better than a single classifier.

Chapter 3

Related Work

For this thesis, three areas of research are particularly relevant: First, sentiment analysis as the field of study to analyze opinions and sentiments in natural language texts is discussed since the opinion of tweets has to be analyzed. Moreover, factorization machines as well as ensemble methods are relevant because these are the two families of classification algorithms used in this thesis. In the first section of this chapter, relevant work in the field of sentiment analysis and the analysis of tweets in particular is presented. The literature research on sentiment analysis relies on the literature survey of Pang and Lee [PL08] and Liu [Liu12] in their comprehensive books about sentiment analysis and opinion mining. The second part will give an overview of current applications of factorization machines and ensemble methods as classifiers.

3.1 Twitter Sentiment Analysis

Sentiment analysis has become a growing field of research in natural language processing. Besides some early work [Car79, WB83] which tried to model human understanding and beliefs, research in sentiment analysis started around the year 2000 due to the growth of user generated content like online reviews and blogs. In 2001, Das and Chen [DC01] proposed a methodology to extract sentiment from stock messages and classify them into optimistic, pessimistic and neutral messages. They used a suite of five different classifiers together with a majority voting schema to predict the final sentiment of the financial chat board messages and achieved an accuracy of 62%. Pang et al. [PLV02] compared the performance of three standard machine learning algorithms (naive Bayes, maximum entropy and SVM) based on their applicability for sentiment classification of movie reviews. They considered the sentiment classification of movie reviews as a binary classification task with a review belonging either to the positive or negative class. The best accuracy of 82.9% has been achieved through a SVM with unigram features. Although these papers used the terms *sentiment* and *opinion* frequently, according to Pang and Lee [PL08] and Liu [Liu12], the phrase *sentiment analysis* first appeared in the paper *Sentiment Analysis: Capturing Favorability Using Natural Language Processing* by Nasukawa and Yi [NY03].

The popularity of Twitter has yielded to some interesting works in recent years. Go et al. [GBH09] were one of the first who used sentiment analysis on Twitter data. As little or no labeled tweets for training and testing were available, they had to build their own datasets. By using happy emoticons (`:)`, `:-)`, `:D`, ...) and sad emoticons (`:(`, `:-(`, `:(`) as query parameters for the public Twitter API[1], they automatically collected tweets with positive sentiment (containing happy emoticons) and negative sentiment (containing sad emoticons). Similar to Pang et al. [PLV02], they tested naive Bayes, maximum entropy and support vector machines as classifiers. Four different kinds of features have been chosen and compared: unigrams, bigrams, the combination of unigrams and bigrams and unigrams with POS tags. The overall best accuracy (83.0%) has been reported when using unigrams and bigrams together with the maximum entropy classifier, whereas SVM with simple unigrams achieved a similar accuracy of 82.2%.

Barbosa and Feng [BF10] proposed a two-step classifier for sentiment detection within tweets. First, a tweet is either classified as subjective or objective (subjectivity detection). Each subjective tweet is then further identified as a tweet with a positive or negative sentiment (polarity detection). Barbosa and Feng used a slighty different approach than Go et al. [GBH09] for building their tweet datasets: Instead of querying the Twitter API directly, they submitted queries to three different real-time sentiment detection websites to get roughly 350,000 tweets (200,000 after cleansing) along with their sentiment (positive, negative or neutral). In the context of feature space, they also tried a new approach. As tweets are always very short, the authors decided against the common use of raw word representation (n-grams) and developed an abstract representation of a tweet, by using two sets of features. A *meta-feature* set consisting of the given word's part of speech, its subjectivity (weak or strong subjectivity) and polarity (positive, negative or neutral), each of them looked up in a dictionary. The second type of features are called *tweet syntax features* and exploit the syntax of a tweet by counting the occurrences of retweets, hashtags, replies, links, punctuations, emoticons and words with upper case letters. All in all they represent a tweet with only 20 features and thus achieved a reasonable performance when training a single SVM with a small number of training tweets (20% and 23.8% error rate with only 2,000 training tweets for the sentiment de-

[1]`https://dev.twitter.com/rest/public` - accessed on 19 August, 2017

tection and polarity detection classifier, respectively). When using a higher number of training tweets and three SVMs to classify a tweet, they reported an error rate of only 18.1% for the subjectivity detection and 18.7% for detecting polarity (positive or negative).
Pak and Paroubek's work [PP10] is similar to Go et al. [GBH09] with the difference that they incorporate neutral/objective tweets as a third class of possible sentiment and put an additional focus on the linguistic analysis of the collected corpora by checking the distribution of part-of-speech tags in subjective (positive or negative) and objective (neutral) tweets. They used the same procedure as Go et al. [GBH09] to collect positive and negative tweets by querying the Twitter API for happy and sad emoticons. They queried Twitter accounts of popular newspapers like the New York Times and Washington Post to obtain objective tweets. In their linguistic analysis of the collected corpora, they observed that some parts of speech, like nouns and verbs in the third person are more common in objective texts and others are more often used in subjective texts, e.g., utterances and personal pronouns. They experimented with uni-, bi- and trigram features and employed naive Bayes, SVM and conditional random fields (CRF) as classifiers. To improve accuracy, common n-grams were discarded by computing the entropy and a newly developed metric called salience was introduced for each n-gram. Salience is an alternative metric to entropy which is based on the probability of a n-gram having a given sentiment. The best accuracy of 61% was reported when using naive Bayes with bigrams.
Similar to the Twitter-related papers referenced so far, Kouloumpis et al. [KTM11] built their own Twitter corpus and used it together with two other corpora. Their training corpus is based on a subset of the Edinburgh Twitter Corpus [POL10]. The emoticon corpus of Go et al. [GBH09] is used to enhance the training corpus while a hand-annotated corpus of approximately 4,000 tweets serves for evaluation. After removing stopwords and normalizing abbreviations, all-caps and character repetitions, they used unigrams, bigrams, the MPQA subjectivity lexicon [WWH05], POS tags and microblogging features (emoticons, abbreviations and intensifiers) as their feature set. An AdaBoost classifier trained on all features except POS tags outperformed SVMs with a best accuracy of 75%. Using all features including POS tags decreased the accuracy of the classifier by 3%.
Agarwal et al. [AXV$^+$11] focused on two sentiment classification tasks: binary classification into negative and positive and three-way classification into positive, neutral and negative sentiment. For each task, they experimented with three types of models. A simple unigram model, a feature based model and a tree kernel based model. The feature based model contains standard features like POS tags, emoticons and word polarity and the kernel based model uses a tree representation of tweets

and calculates the similarity between trees to deduce the sentiment. All models use support vector machines as the classification algorithm. The authors used a different approach to obtain their data. Instead of using noisy labels like emoticons [GBH09] or hashtags [KTM11] to retrieve and annotate English tweets, the authors used an existing manually annotated Twitter corpus containing roughly 12,000 tweets, originally in various languages, which have been translated into English using Google Translate. After sanitizing and balancing the dataset, 5,127 tweets remain for training and testing. Their experiments showed that their feature based model performs similar to the unigram model whereas the tree kernel based model outperforms both. Best accuracies were reported when enhancing the feature model with unigrams (75.36%) and combining the feature model with the tree based model (60.83%) for the binary task and three-way task, respectively.
Martin Illecker [Ill15] and Zangerle et al. [ZIS16] presented an approach called SentiStorm, to classify tweets within a real-time processing system. They used the datasets from the SemEval-2013 challenge to train and evaluate their system. After data preprocessing using regular expressions, the tf-idf schema, POS tags and sentiment lexicons were used to generate the feature vectors. A SVM with a radial basis function (RBF) kernel was used to classify each tweet according to a three-way (positive, negative and neutral) classification schema. SentiStorm achieved an accuracy of 70.21%, which makes its performance comparable with other systems of the SemEval challenge.

3.2 Applications of Factorization Machines and Ensemble Methods

Below we present academic work that make use of factorization machines or ensemble methods in several machine learning tasks.

Factorization Machines

As factorization machines (FM) are a relatively new topic in machine learning, there are fewer applications compared to the typical learning algorithms like support vector machines or naive Bayes. FMs are a general machine learning algorithm that uses factorization to improve prediction quality under high data sparsity under linear complexity. Therefore, FMs tend to be appropriate for text classification, especially for short texts like tweets as they consist only of a few words.
Steffen Rendle [Ren10] presented factorization machines as a new learning algorithm that can be compared with SVMs but addresses some of their problems: Unlike SVMs, factorization machines perform well on

very sparse data and can be calculated in linear time instead of polynomial time (in case of SVMs with a polynomial kernel).
The earliest use of FMs are presented in [RGFST11] and [FSTR11]. Both papers demonstrate the advantages of FMs over traditional models. Rendle et al. [RGFST11] presented how to use FMs for context-aware recommender systems. Context-aware rating predictors use, in addition to user and items, contextual information about the situation in which the rating of the item happens (e.g., time, mood, location or weather). This additional information is referred to as context and can provide better recommendations than non-context-aware systems [KABO10]. The authors of [RGFST11] showed that their FM implementation can achieve better results in terms of faster runtime and prediction quality compared to other best performing methods in context-aware recommendation.
Freudenthaler et al. [FSTR11] proposed an extension to FMs by using structured Bayesian inference. Their Bayesian Factorization Machine (BFM) performed better on the Netflix challenge compared to state-of-the-art methods like stochastic gradient descent (SGD) and Bayesian probabilistic matrix factorization (BPMF).
To the best of our knowledge, one of the first papers which use factorization machines in combination with Twitter data are Hong et al. [HDD13] and Qiang et al. [QLY13]. The approach of Hong et al. [HDD13] predicts user decisions (if a user retweets a certain tweet) and models content (what are the users' interests) simultaneously by analyzing their Twitter dataset consisting of 27 million tweets. Therefore, they proposed Co-Factorization Machines (CoFM), an extension to FMs which can handle multiple aspects (decisions and content) of the same tweet where each aspect is represented in a separate FM. The authors reported that their CoFM system performs significantly better than state-of-the-art baselines like matrix factorization and standard FMs.
To rank the most recent tweets according to the user's relevance (real-time search), Qiang et al. [QLY13] proposed Ranking Factorization Machines to model microblog ranking with factorization machines. They used procedures like query expansion and extracting HTML from posted URLs to enhance their set of features used to model a tweet. Experiments on the Tweet11 corpus of the Twentieth Text REtrival Conference (TREC 2011) showed that their Ranking FM approach achieves better results than several baseline methods, including the winner of the TREC 2012 real-time search task [HLY$^+$12].
Oentaryo et al. [OLL$^+$14] developed Hierarchical Importance-aware Factorization Machines (HIFM). HIFMs are an extension to factorization machines which include importance weights and hierarchical learning into the generic FM framework. Oentaryo et al. reported that these enhancements improve the prediction of dynamic advertising responses (i.e., estimating click-through rate to maximize advertising revenue) by

overcoming the cold-start problem and weighting more important ads higher.
Chen et al. [CDCX14] use factorization machines to predict the trend of stock data. Unlike traditional stock market predictions which are based on the analysis of historical stock data, they incorporate social media to predict if a given stock price rises or drops at the end of the day. They used user messages from Sina Weibo, a popular Chinese microblogging website and experimented with different textual representations and showed that their approach using FMs achieves 81% accuracy and thus performs better than other baseline approaches.

Ensemble Methods

Ensemble methods train multiple instances of learners on individual data and combine their output to predict more accurate results. Ensemble methods have a wide range of applications including image retrieval [JED04], face recognition [HZZT00], collaborative filtering [Kor09] or data mining [NMPS$^+$09]. Hence, we present papers making use of ensembles only in the context of sentiment analysis in the following paragraphs.
[BF10] and [KTM11] presented in Section 3.1 made already use of ensemble methods. Barbosa and Feng [BF10] did not use accuracy to measure the performance of their classifier. Instead, they measured their results with an error rate metric that has not been defined by the authors. The lowest error rate in polarity detection (18.7%) has been achieved when using an ensemble of three SVMs and taking the prediction of the SVM with the highest confidence as the final prediction value. Kouloumpis et al. [KTM11] used an AdaBoost algorithm with 500 rounds of boosting in combination with cross validation for their experiments.
Wilson et al. [WWH05] built a two-step classification system to distinguish the polarity of phrases. Similar to Barbosa and Feng [BF10], they distinguished between neutral and polar phrases in the first step. For the second step they tried to classify each polar phrase as positive, neutral, both or neutral. Although neutral phrases were marked as nonpolar in the first step, the authors noted that some neutral expressions are classified as polar and therefore neutral has to be considered as a separate class in the second step. In both classification steps, they used an AdaBoost learning algorithm with 5,000 rounds of boosting. The first classifier (neutral-polar classification) achieved 75.9% accuracy on their test set while the second classifier (four-way polarity classification) achieved an accuracy of 65.7%.
Wan [Wan08] developed a new unsupervised learning approach to classify the sentiment of non-English resources. First, the author translated Chinese reviews into English by using various machine translation ser-

vices to obtain different English versions of the same Chinese review. By using different Chinese and English sentiment lexicons, semantic orientation values (a positive/negative value represents a positive/negative review) for every version of the review were computed. To aggregate the set of orientation values to a final polarity, seven simple ensemble methods (e.g., average, weighted average, majority voting) were used. Wan reported to achieve the best accuracy of 86.1% when using a weighted average of the semantic orientation values of two English translations and the original Chinese review.

Martínez-Cámera et al. [MCGVF$^+$13] built an ensemble classifier for polarity classification of Spanish tweets. The ensemble combines the output of three different polarity classifiers: a SVM which uses a generated sentiment lexicon as features, a vector model generated with a deep learning algorithm and an unsupervised learning method using three sentiment lexicons. The voting scheme to combine the outputs is a simple majority voting with a tie resolution strategy which considers results with a tie as neutral output. The authors stated that their ensemble improves the performance of the three base classifiers.

Chapter 4

SentTwi

The following chapter describes our approach to build *SentTwi*, a tool for sentiment detection of tweets. A pipeline is used to process and transform the input tweets step by step into applicable input for the classifier. The classification program is written in *Python 3* and uses the popular machine learning library *scikit-learn*[1]. The pipeline consists of five main steps:

1. Read Tweets
2. Preprocess Tweets
3. POS Tagging
4. Feature Generation
5. Classification

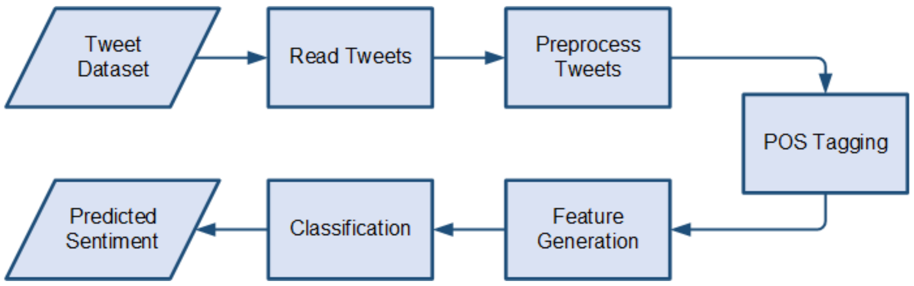

Figure 4.1: Classification Pipeline Consisting of Five Consecutive Steps.

Figure 4.1 visualizes this learning procedure which starts with the process of importing the tweets and returns the predicted sentiment of the test tweets at the end. The first step reads the training and test tweets from a file and the second step preprocesses them. During the preprocessing, each tweet gets normalized by converting all characters into

[1]`http://scikit-learn.org/` - accessed on 19 August, 2017

lowercase and simplifying negations, abbreviations and slang. The *POS Tagger* component tokenizes each tweet into single tokens and assigns each token a part-of-speech tag. The *feature generation* component extracts and generates the features using the transformed tweets, their POS tags and various sentiment lexicons and outputs the feature vectors for the classifier. Finally, the classifier component uses the features generated in the previous step and the input labels given from the first component to predict the sentiment of the test tweets. The classifier can be one of the classification algorithms presented in Section 2.4 and 2.5: *factorization machine*, *AdaBoost* or *Bagging*. Since we have a training and a test set, we need to extract features from both datasets. Therefore, all steps except the last classification step are applied on both corpora. Each of the five steps will be described in detail in the remaining chapter.

4.1 Read Tweets

The first component reads the tweets from a given TSV file. A TSV (tab-separated values) file is similar to a CSV file, a simple text file where each data record consists of a single line separated by a tab. This *TweetReader* expects the same tab separated format used in the *SemEval 2016 Task 4 Subtask A* dataset files[2]. The training and test files provided by the SemEval organizers are formatted as follows:

```
id        label
```

where `id` refers to the tweet's unique ID in the Twitter API and `label` is the annotated sentiment and can be either *positive*, *neutral* or *negative*. To protect the privacy of the Twitter user and prevent abuse of the tweets, the provided corpus does not contain the actual tweet. To obtain the desired tweets, the SemEval organizers provide a download script[3] to crawl these tweets via the official Twitter API. After downloading the tweets, the format of the TSV file looks as follows:

```
id        label        tweet
```

The TweetReader parses each data record line by line into memory and stores them as a list of tweets and a list of labels. To use the sentiment labels as input for the classifier algorithm, the textual labels *positive*, *neutral* and *negative* are mapped to their numeric representation of `1`, `0` and `-1`, respectively. This importing process is executed two times on different data: First the training tweets get imported and then the test tweets are read.

[2]`http://alt.qcri.org/semeval2016/task4` - accessed on 19 August, 2017
[3]`https://github.com/aritter/twitter_download` - accessed on 19 August, 2017

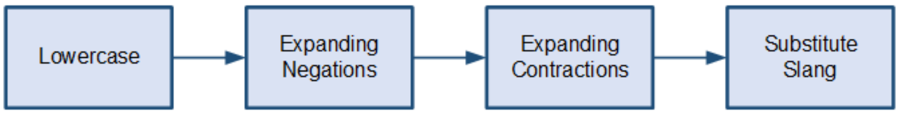

Figure 4.2: Preprocessing Steps

4.2 Preprocessor

The *preprocessor* is the first component that processes the actual content of the tweets. Due to the limited length of tweets, users sometimes use linguistic shortenings like contractions ("I'm", "it's" or "y'all"), abbreviations ("b4" or "pic") and acronyms ("AFAIK", "ICYMI" or "TLDR") to not exceed the 140 character limit. The preprocessor aims to turn these shortenings back into their normal form since most information retrieval techniques require standard English text to work. This also decreases the number of possible phrases in a tweet and thus, reduces the complexity of the sentences and feature size. The preprocessing phase is also split into multiple small sub-steps. Figure 4.2 illustrates this workflow, which is described further in the following.

Lowercase The first operation turns the whole tweet into *lowercase* text to remove case sensitivity and damp the emphasis of all-caps words like "NEVER", "STOP" or "THANKS". Although there exists alternatives like *truecasing* [LIRK03], which tries to correct capitalization, all words are converted to lowercase due to the informal language used in tweets. Similar to short messaging and instant messaging, Twitter users tend to write lowercase regardless of the correct capitalization or capitalize single words to indicate importance. Therefore, correct truecasing can be a challenging task with unknown benefit and lowercasing everything is a simple and practical solution [MRS08].

Negation *Substitute negations* expands contractions of negations like "doesn't" and "wouldn't" to their written base form "do not" and "will not". Using the base form rather than keeping the tense and person of the negation reduces the complexity as "ain't" and "weren't" get both substituted by "is not".

Contraction The third step is similar to the previous one. Miscellaneous contractions like "you're" and "it's" are substituted by their written base form "you are" and "it is". The goal of these substitutions is the same as before: reducing the feature size and normalizing tweets.

Slang	**Meaning**
'cause	because
c'mon	come on
omg	oh my god
u	you
y'all	you all
yr	year

Table 4.1: Examples of Slang Abbreviations and Acronyms with their Corresponding Substitution

Step	**Unprocessed Tweet**	**Preprocessed Tweet**
Case Folding	**This** is **AWESOME**!!!	**this** is **awesome**!!!
Negations	I **won't** be in class tomorrow	i **will not** be in class tomorrow
Contractions	**It's** cold. **I'm** freezing :(	**it is** cold. **i am** freezing :(
Slang	**OMG**! **U** are crazy!	**oh my god**! **you** are crazy!

Table 4.2: Example Tweets Demonstrating Each of the Four Preprocessing Steps. Text in **Bold** Indicates the Changes Made During the Respective Step.

Slang In the last step, common slang abbreviations and acronyms are substituted with their meaning. Table 4.1 shows some slang examples and the meaning they get substituted with. Substituting slang phrases with standard English is needed to allow later components like the POS tagger and the sentiment lexicons to process these phrases correctly.

To recap the preprocessor steps, tweets are turned into all lowercase words first and then contractions and slang abbreviations are expanded to reduce the sparsity of the feature vector by unifying similar expressions. Table 4.2 illustrates these preprocessing steps and lists example tweets returned by the TweetReader before and after the preprocessor phase.

4.3 POS Tagging

The POS tagger receives the preprocessed tweets from the *Preprocessor* and computes the corresponding part-of-speech tags. Popular POS tagger software like the Stanford Tagger [TKMS03] work well on classical text data (articles, books, etc.) but are not suited for tweets or similar user generated content [GSO$^+$11]. Specialized taggers are trained on tweets to be able to detect tokens often used in tweets and social media like emoticons, hashtags and mentions. In literature, there are two relevant POS taggers available to tag tweets: first, the *CMU ARK Tagger* by Owoputi et al. [OOD$^+$13] and second, the *GATE Tagger* proposed by Derczynski et al. [DRCB13].

The ARK Tagger has been developed at the Carnegie Mellon University and consists of a Java-based tokenizer and POS tagger. The tagger is trained on a manually labeled set of tweets and uses the POS tagset by Gimpel et al. [GSO$^+$11] consisting of 25 part-of-speech tags which can be found in Appendix A.1. The ARK tagger uses a first-order Maximum Entropy Markov model (MEMM) as the tagging model which allows efficient training and decoding (i.e., finding the most likely sequence of tags for a given tweet) and unsupervised word clusters combined with a hidden Markov model (HMM) to improve performance further.

The GATE Twitter POS tagger[4] is part of GATE Developer[5], a Java development environment for human language processing. The GATE tagger is available as a Java plugin for the GATE software, a standalone Java program or as a trained model for the Stanford Tagger. All versions use an improved Twitter model which addresses errors frequently made by other POS taggers. The GATE team mentions slang, unknown words and misspellings as common mistakes. For compatibility with possible existing text processing tools, the GATE tagger uses the Penn Treebank tagset [MMS93]. The Penn Treebank tagset is the de facto standard tagset for English texts and consists of 36 different tags, which is a slightly more complex tagset, compared to Gimpel's custom tagging scheme used in the ARK tagger.

Both teams behind the two Twitter POS taggers report a similarly high accuracy of 93.2% and 88% for the ARK tagger and the GATE tagger, respectively. Our reason for the decision to use the ARK tagger is primarily based on Illecker's experience [Ill15] with the two presented taggers and the slightly better accuracy of the ARK tagger. Illecker performed an experiment to determine the more suitable tagger for his approach. He compared the standalone performance of the two taggers by the number of tweets they can tag per second and the number of parallel threads they are running on and reported an increase of 20% to 30% in the number of tagged tweets per second when using the ARK tagger in place of the GATE tagger.

[4]`https://gate.ac.uk/wiki/twitter-postagger.html` - accessed on 19 August, 2017

[5]`https://gate.ac.uk/` - accessed on 19 August, 2017

Preprocessed Tweet	**Tagger Output**
@microsoft i will be downgrading and let #windows10 be out for almost the 1st yr b4 trying it again :/ #windows10fail	[('@microsoft', @, 0.9989), ('i', O, 0.9883), ('will', V, 0.9997), ('be', V, 0.9994), ('downgrading', V, 0.9715), ('and', &, 0.9965), ('let', V, 0.98), ('#windows10', #, 0.9686), ('be', V, 0.998), ('out', T, 0.6579), ('for', P, 0.997), ('almost', R, 0.9203), ('the', D, 0.9974), ('1st', A, 0.8932), ('yr', N, 0.9555), ('b4', P, 0.9489), ('trying', V, 0.9996), ('it', O, 0.9902), ('again', R, 0.9877), (':/', E, 0.9551), ('#windows10fail', #, 0.9361)]

Table 4.3: Example Tweet and List of the Corresponding POS Tags as Tuples (Token, Tag, Confidence)

Table 4.3 shows a preprocessed example tweet and the corresponding output tags of the ARK tagger. For each token, a tuple containing the *word*, *tag* and the *confidence* of the assigned tag is returned. We observe that for the example tweet the tagger has a confidence of at least 90%. The lower confidence of *out* with 0.6579 can be explained since *out* can be either tagged as a verb particle T (as in this example) or as a preposition P (as in "it blazed out into space"). Also Twitter-specific tokens like the at-mention "@microsoft" and the hashtags "#windows10" and "#windows10fail" are correctly annotated with @ and #. Similarly, the ARK POS tagger recognizes abbreviations ("1st", "yr" and "b4") and emoticons with a quite high confidence.

After the POS tagging process, a small post-processing component allows to prune the token output of the tagger. Since tokens tagged with U (URL or email address) normally do not convey any sentiment as they are often shortened or random links, they are dropped from the list of tokens. The same is done with stopwords since they have little semantic content and would only introduce unwanted noise.

The POS Tagging component receives a list of tweets and returns a list of tuples consisting of the token, the assigned tag and the assignment confidence for each tweet. This tags are then used as input in various parts of the following feature generation component.

4.4 Features

Generating high-quality features is the most crucial part in the classification process. This component extracts a variety of features from two input sources and generates a single feature matrix for the classifier. Features typically used in sentiment analysis such as the bag-of-words model, POS features and sentiment lexicons are extracted. The *Feature Generation component* is composed of six sub-components which independently receive either the preprocessed raw tweets from the preprocessor or the tagged tweets annotated by the POS tagger. These six feature vectors are then concatenated end-to-end into the final feature matrix. The sub-components are presented in the subsequent pages.

4.4.1 Bag-of-Words

The first sub-component of the feature extraction procedure uses the *bag-of-words model* [MRS08] which ignores the word order and turns the preprocessed raw tweets into a feature matrix. This process is also known as *vectorization* and thus, this feature extraction component is called *vectorizer*. One of the two different weightings presented in Section 2.2 can be used to describe the features: either a simple *term frequency* (*tf*) weighting which counts the occurrences of each word in the whole corpora or the more advanced *term frequency-inverse document frequency* (*tf-idf*) weighting.
First of all, the raw tweet string has to be split into single word tokens. Although the POS tagger has already tokenized the string during its tagging process, we want to use a second tokenizer for one simple reason. Using a dedicated tokenizer for the bag-of-words features allows us to replace this tokenizer without touching the POS part of the system since the CMU ARK tagger depends on its own built-in tokenizer. Because this tokenizer performs well on tweets, the ARK team provides a self-contained version of the tagger's tokenizer, called *Twokenize*[6], which is used as the tokenizer for the bag-of-word features.
After tokenizing the tweet, another post-processing step is done to remove unwanted tokens or modify specific words. Analogously to the post-processing step in the POS tagging process, which drops tokens with little semantic value from the POS list, a similar post-process removes stopwords, URLs and punctuation from the bag-of-words feature set. User mentions and hashtags are trimmed by removing their first character (i.e., "@" and "#") to combine the features of these Twitter tokens and normal words, e.g., "#married" contributes now to the feature "married" instead of representing a new one. The last step in this post-processing uses the Snowball stemmer [Por01] to apply stemming

[6]`http://www.cs.cmu.edu/~ark/TweetNLP/` - accessed on 19 August, 2017

Group	**POS Tag**
Nouns	**N**, O, ^, S, Z
Verbs	**V**
Adjectives	**A**
Adverbs	**R**
Interjections	**!**
Twitter	**#**, @, ~, U
Emoticons	**E**

Table 4.4: Mapping from ARK POS Tags to Features. Tags in **Bold** are Used as Feature Names in the Feature Vector.

to each remaining token which reduces the words to their common word stem.
The feature vector generated by the vectorizer is the largest feature vector consisting of tens of thousands to multiple hundred thousands of features, depending on the size of the training corpora and the settings of the vectorizer.

4.4.2 POS Tags

The POS tags generated by the POS Tagging component from Section 4.3 have to be transformed into a numerical representation. The ARK tagger assigns each token one tag from the Gimpel tagset (Appendix A.1) consisting of 25 part-of-speech tags. Similar to Zhu et al. [ZKM14], we count the occurrences of groups of tags instead including each assigned token in the feature vector individually. Table 4.4 enumerates the seven tag groups and the corresponding POS tags contributing to their counts. Tags written in **bold** are the group's abbreviation used as the feature name. Other tags like determiners (D) and prepositions (P) are ignored as they usually have only grammatical functions and have little to no semantic content.
The counts of each group are normalized by the length of tokens to scale each feature to the interval [0, 1]. Table 4.5 shows the feature vector obtained from the *ARK POS tagging component* for the example tweet from Table 4.3. It can be seen that some elements are omitted since the post-processing step has removed useless tokens. The remaining list of tokens contains ten elements with three Twitter tags (one @ and two # tags) which yields to a normalized value of $\frac{3}{10} = 0.3$. If features are not present in a single feature vector (in this example, adverbs/R and interjections/!), their value is automatically set to zero in the overall feature matrix.

POS Tagger Output	**Feature Vector**				
	N	**V**	**A**	**#**	**E**
[('@microsoft', @), ('downgrading', V), ('let', V), ('#windows10', #), ('1st', A), ('yr', N), ('b4', P), ('trying', V), (':/', E), ('#windows10fail', #)]	[0.1,	0.3,	0.1,	0.3,	0.1]

Table 4.5: Feature Vector of the Post-Processed POS Example Tweet Found in Table 4.3. *(Confidence Values are Elided)*

4.4.3 Punctuation

Some punctuation, especially exclamation marks, can intensify positive as well as negative sentiment. Along the lines of Zhu et al. [ZKM14], four different punctuation features are extracted for each tweet:

- Number of sequences of continuous *exclamation marks*
- Number of sequences of continuous *question marks*
- Number of sequences of continuous *exclamation* or *question marks* (e.g., !?, !!!???!!!)
- Whether the *last token* of the tweet contains either an *exclamation* or *question mark*

The calculation can easily be done by matching each token against a simple regular expression for each feature.

4.4.4 Emoticons

Analogous to the punctuation features, a set of emoticon features is used to detect sentiment beyond words. The occurrences of emoticons in the tweets are counted and each emoticon is classified either as *positive*, *neutral* or *negative*. For each sentiment a list of emoticons has been manually picked from various sources [7] [8] and extended. The three lists contain 91 positive, 34 neutral and 66 negative emoticons, respectively, and can be found in Appendix A.2.

4.4.5 Sentiment Lexicons

Multiple sentiment lexicons are used to determine the sentiment of single tokens. A sentiment lexicon is a list of words which associates each word a sentiment score. There exist various ways to assign a word a polarity.

[7] `http://cool-smileys.com/text-emoticons` - accessed on 19 August, 2017

[8] `https://en.wikipedia.org/wiki/List_of_emoticons` - accessed on 19 August, 2017

Most lexicons use a positive/negative scale to denote the intensity of the sentiment. Some more specialized lexicons use category values like *strong* and *weak* or *anger*, *sadness* and *joy* to associate implications and emotions with words and phrases. There exist even lexicons which associate colors with target words [Moh11]. However, *SentTwi* uses only lexicons with categorical (*positive*, *negative*, *neutral*) or numerical target scores (e.g., $\{-1, 0, 1\}$, $[0, 1]$, $[-5, 5]$). To be able to compare and combine the results of the lexicons, all scores are scaled to an interval of $[0, 1]$ using *Min-Max scaling* based on the following equation:

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

where x is the actual sentiment score from the lexicon and the two extrema are the maximal and minimal sentiment scores found in the lexicon.
Seven different lexicons are used and for six of them, feature vectors consisting of the following six features are built:

- Number of words with a *positive* sentiment
- Number of words with a *negative* sentiment
- Number of words with any sentiment assigned
- *Sum* of all sentiment scores
- Maximum *positive* sentiment
- Maximum *negative* sentiment

For each tweet, all six features are calculated by looking up the sentiment of the tokens in the lexicons. *SentTwi* uses the same boundaries for positive, neutral and negative sentiment as proposed by Illecker [Ill15]: A rescaled sentiment score < 0.45 is considered as *negative sentiment* and words with a score > 0.55 are treated as words with a *positive sentiment.* Sentiment scores between the two boundaries are *neutral words* and contribute to the third and fourth feature.
The used lexicons are presented and compared in the paragraphs below.

AFINN

The *AFINN-111* lexicon [Nie11] is a list of English words built by Finn Årup Nielsen. The author manually collected and labeled 2477 unique words with an integer score between -5 (negative) and $+5$ (positive). After rescaling, the lexicon contains 878 positive words, 1598 negative words and one neutral word.

MaxDiff Twitter Sentiment Lexicon

The *SemEval-2015 English Twitter Lexicon*, sometimes also referred as *MaxDiff Twitter Lexicon* is a lexicon of single words and two-word negated phrases with scores between -1 and $+1$. It has been developed by Kiritchenko et al. [KZM, RNK+15] at the National Research Council Canada (NRC) in the course of their participation at the *SemEval-2015 Sentiment Analysis in Twitter challenge*. The 1515 different phrases are taken from English Twitter corpora and are manually annotated through crowdsourcing using the MaxDiff annotation method[9]. Since the words are taken from tweets, the lexicon contains informal language characteristics like misspellings and hashtags which makes the lexicon predestined for Twitter sentiment analysis. At the end, the MaxDiff Lexicon consists of 681 positive, 633 negative and 201 neutral terms.

Opinion Lexicon

Liu and Hu built the *Opinion Lexicon* [HL04, LHC05] using words extracted from online customer product reviews. This generated lexicon contains 2006 positive and 4783 negative words. Feature scaling is easily done by mapping positive scores to 1 and negative scores to 0.

Sentiment140 Lexicon

The *Sentiment140 Lexicon* has been created by the NRC Canada team for their SemEval-2013 submission [MKZ13]. The lexicon has been generated based on the *sentiment140* corpus [GBH09] containing 1.6 million tweets and consists of 62,468 unigrams (25,906 positive, 22,481 neutral, 14,081 negative). The team calculated for each word a decimal sentiment score between -5 and $+5$ by calculating the pointwise mutual information (PMI) for terms found in positive and negative tweets.

SentiStrength

SentiStrength[10] is a sentiment-analysis program developed by Thelwall et al. [TBP+10, TBP12]. In our approach, we only use the sentiment terms included in SentiStrength. This sentiment lookup table is a collection of 648 positive terms and 2009 negative terms. Positive terms are assigned integer values from 1 to 5, whereas negative terms get scores from -5 to -1. The special feature of this lexicon is that it does not

[9]Maximum Difference Scaling aka Best-Worst Scaling [LFM15] is a comparative annotation schema which let annotators select the best and worst item out of four possible items.

[10]`http://sentistrength.wlv.ac.uk/` - accessed on 19 August, 2017

SynsetTerms	interactive interactional
Gloss	capable of acting on or influencing each other
Scores (pos/neg/obj)	0.375/0.125/0.5

Table 4.6: SentiWordNet: Synset Composed of the Two Synonymous Terms “Interactive” and “Interactional”

only contains words commonly used in social media but also terms ending with a wildcard. For example, the term “beaut*” would match any word starting with “beaut” (e.g., “beauty”, “beautiful”, “beautifully”). The authors manually assigned and revised the sentiment scores and wildcards to improve the negative sentiment strength detection of the SentiStrength algorithm.

SentiWordNet

SentiWordNet [ES06, BES10] is a generated sentiment lexicon based on *WordNet* synsets. A *synset* is a set of synonyms linked together by similar senses. SentiWordNet assigns each synset three sentiment scores (positive, negative and objective) based on their glosses. Each of the three scores range in the interval $[0.0, 1.0]$ and their sum add up to 1.0. The SentiWordNet lexicon includes almost 207 thousand terms with their POS and over 117 thousand synsets. Table 4.6 shows an example synset for the term “interactive” with the synonym “interactional”. Both SynsetTerms are grouped into a synset with the same meaning.
Since terms with multiple meanings belong to multiple synsets, a single sentiment score for a set of synsets has to be calculated to be able to lookup a sentiment score for a single token. The sentiment score for a term is calculated by a weighted average. After transforming all synsets into single lookups, our SentiWordNet lexicon contains more than 155 thousand terms associated with a sentiment score between -1 and 1.

VADER

VADER Sentiment [HG14] is a lexicon and rule-based sentiment analysis tool. VADER (Valence Aware Dictionary for sEntiment Reasoning) uses a human-curated base lexicon with approx. 7,500 words, including social media terms (emoticons, abbreviations, slang), in combination with a small set of rules to detect sentiment. In contrast to other sentiment lexicons, VADER performs on a sentence level basis for sentiment calculation. For each input string, four scores are calculated: a *positive*, *neutral* and *negative* score between 0 and 1 which add up to 1 and an additional *compound* score. The compound score is the sum of the valence scores for each word normalized to be between -1 and 1. The valence score is internally used by VADER to measure the sentiment intensity of words and ranges from -4 (extremely negative) to +4 (extremely positive). The sentiment scores are modified by analyzing the grammatical structure and looking for booster and shifting words like "extremely", "not" or "but". Compared to the other lexicons, VADER computes for each tweet a feature vector consisting of four sentiment scores instead of returning a vector with six features.

4.5 Classifiers

The last component in the classification pipeline is the *classifier*. The classifier is the core component in the whole classification pipeline process. It uses machine learning to classify new tweets on the basis of a previously learned feature vector produced by the feature generation component. The modular concept of *scikit-learn* allows an easy replacement of the classifier component. *SentTwi* can use either a *factorization machine*, an *AdaBoost* or a *Bagging* classification algorithm to classify tweets. In the following, the configuration of the classifiers presented in Section 2.4 and 2.5 will be explained. For each classifier, a grid search is used to find the parameter configuration with the best performance.

4.5.1 Factorization Machines

Since factorization machines are a comparatively recent machine learning concept, there exist few libraries yet. *LibFm*[11] is the reference implementation for factorization machines and has been proposed by their developer Rendle [Ren12a]. It is fully written in *C++* and is controlled via command line arguments. To use libFM in our project, we make use of a Python wrapper called *pywFM*[12]. This wrapper starts the libFM process with the arguments supplied by our *Python code* and returns the result written by libFM.
Unfortunately, libFM only supports binary classification. To be able to predict our tweets according to the three-class schema (positive, negative, neutral), we could either use multiclass classification (One-vs-Rest or One-vs-One) [MRS08] or compute regression values instead of classes and map these values to our three classes. Although scikit-learn offers a multiclass module to perform One-vs-Rest and One-vs-One classifications, this approach would have implied additional efforts since pywFM does not conform with scikit-learns interface, meaning an own multiclass classification implementation would have to be written. Therefore,

[11]`http://www.libfm.org/` - accessed on 19 August, 2017
[12]`https://github.com/jfloff/pywFM` - accessed on 19 August, 2017

we decided to perform regression instead of classification to obtain real-valued scores between -1 and 1 and map each regression score s to a sentiment class c according to the following equation:

$$c = \begin{cases} \text{positive} & \text{if } s > 0.4 \\ \text{negative} & \text{if } s < -0.4 \\ \text{neutral} & \text{else} \end{cases} \tag{4.2}$$

The interval for the neutral class is larger than the interval used to map the sentiment of the sentiment lexicon features ($[-0.4, 0.4]$ versus $[0.45, 0.55]$), as first attempts using FMs with regression indicate that using a larger interval improves the prediction quality.
Factorization machines allow three different learning methods (see Section 2.4.1): *Stochastic Gradient Descent* (SGD), *Alternating Least Square* (ALS) and *Markov Chain Monte Carlo Inference* (MCMC). We use MCMC, since it is the recommended method by Rendle [Ren12a] as it has the fewest input parameters which reduces the time to find the best parameters during the grid search process. LibFM and thus pywFM too, needs five parameters as command line arguments. The three parameters `k0`, `k1` and `k2` determine how the model parameters Θ (w_0, $\mathbf{w}$ and $\mathbf{V}$) are used. To recap: w_0 defines the global bias, $\mathbf{w}$ are the unary interactions and $\mathbf{V}$ models the pairwise interactions between the variables. The libFM parameters `k0` and `k1` state whether w_0 and $\mathbf{w}$ should be used or not, while `k2` sets the dimensionality of the factorization parameter (i.e., hyperparameter k in $\mathbf{V} \in \mathbb{R}^{n \times k}$). The number of iterations to find the regularization values with Gibbs sampling are restricted by the fourth parameter `iter`. The last parameter `init_stdev` is the initialization hyperparameter which initializes the model parameter $\mathbf{V}$. In the MCMC learning model, this parameter can be set to 0, whereas appropriate values can speed up the learning process.

4.5.2 AdaBoost

AdaBoost is the first of the two ensemble methods used in this thesis. AdaBoost is a boosting algorithm which chooses the training examples in each boosting round based on the classification results of previous rounds. Scikit-learn provides a module called *sklearn.ensemble* which contains ensemble methods for tasks like classification and regression. The `AdaBoostClassifier`[13] requires three parameters. The parameter `n_estimators` specifies the size of the ensemble and the `learning_rate` parameter controls the contribution of each classifier in the final output.

[13]`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html` - accessed on 19 August, 2017

The last parameter `base_estimator` defines the type of classifier the ensemble consists of and defaults to a decision tree.

4.5.3 Bagging

The Bagging classifier is our second ensemble method and is similar to AdaBoost in terms of usage and parameter tuning. Each Bagging classifier chooses the training instances randomly from the given training set. In addition to the parameters found in AdaBoost, scikit-learn's `BaggingClassifier`[14] allows to specify the number of training samples each classifier selects (`max_samples`) and whether these samples should be drawn with replacement (`boostrap`). As an alternative option to the bootstrapping algorithm based on the sample size, we can split the input data for each classifier feature-based, which is known as *random subspace* [Ho98]. Classifiers using random subspace employ all samples with a subset of features instead of a subset of samples with all features. This option introduces two additional parameters, `max_features` and `boostrap_features`, which set the feature subset size for each classifier and whether features are drawn with replacement, respectively. If both parameters (`max_samples` and `max_features`) are used, this method is known as *Random Patches* [LG12].

4.6 Grid Search

Almost every component presented so far allows to specify parameters to tune their behavior. A *grid search* is used to find the optimal set of parameters to achieve the best performance on unseen data. This is done by performing an exhaustive search through a manually preselected subset of all possible parameter combinations.
Since a grid search with the complete SemEval dataset presented in Table 2.5 would be too time-consuming, only the SemEval 2016 train and development datasets are used. To avoid overfitting, an *n-fold cross-validation* schema is used which divides the input dataset into n equally sized subsets. The grid search executes each parameter configuration n-times with $n-1$ subsets as training data and the remaining n–th subset as test set. Since the SemEval data is not equally distributed among the three sentiment values *positive*, *neutral* and *negative*, a variation of n-fold, called *stratified n-fold* is used. Stratified n-fold splits the data into n subsets as well, but preserves the percentage of samples in each class.

[14] `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html` - accessed on 19 August, 2017

Although scikit-learn offers a convenient `GridSearchCV` class which runs a grid search using cross-validation, an own grid search implementation with cross-validation had to be written because *pywFM* does not conform to the interface required by GridSearchCV.
To find the optimal choice of parameters, two different grid searches have been executed. Since each feature component can be parameterized and the classifiers allow various settings, performing one single grid search over the whole parameter space would be too exhaustive and unfeasible. Both grid searches use 5-fold cross-validation to determine the settings with the highest accuracy score.

4.6.1 Classifier Grid Search

The first grid search tries to find appropriate parameters for the classifier. This grid search is used to determine the set of parameters which maximize the accuracy score while using the default settings for the feature pipeline. The parameters found by this search are then used in the second grid search to find the best parameters for the feature vector components. Since each classifier has a different parameter space (i.e., differ in parameter names, amount of parameters and allowed values), this grid search has to be executed for each classifier separately. Therefore, the *Classifier Grid Search* has to try 512, 150 and 1024 different parameter combinations for the factorization machine, AdaBoost and Bagging classifier, respectively.

4.6.2 Feature Vector Grid Search

After the first grid search has found the optimal parameters for each of the three classifier types, the *Feature Vector Grid Search* tries to identify the unknown optimal parameters of the pipeline component. Similar to the first grid search, this grid search uses 5-fold cross-validation on the same SemEval 2016 dataset. In the end, the *Feature Vector Grid Search* component has tried for each of the three classifiers 6144 different combinations of the eight parameters listed in Table 4.7. The set of parameters and their optimal value found by both grid searches are displayed in the following tables. Tables 4.8, 4.9 and 4.10 show the values found by the first grid search and Table 4.11 presents the set of pipeline parameters for each of the three pipelines together with the accuracy achieved with the parameters found by both grid searches.

Parameter	**Description**
`vectorizer`	Specifies the type of vectorizer: - `CountVectorizer` uses *tf weighting* - `TfIdfVectorizer` uses *tf-idf weighting*
`pre_active`	Whether the preprocessor should transform the tweets at first (Section 4.2).
`stop_words`	Whether the vectorizer should remove stopwords during tokenization.
`binary`	If `true`, the vectorizer considers the count of words in each tweet as binary instead as an integer (i.e., term presence instead term frequency)
`ngram_range`	Specifies the lower and upper bound of the n-gram size generated by the vectorizer. E.g., the value `(1,3)` would consider uni-, bi- and trigrams.
`max_features`	The number of features the vectorizer will extract at most, ordered by their frequency.
`min_df`	Ignores terms that have a lower *absolute document frequency* (number of tweets) than this value.
`max_df`	Ignores terms that have a higher *relative document frequency* (percentage of tweets) than this value.

Table 4.7: Parameter Space of the Feature Vector Grid Search

Parameter	**Optimal Value**
`init_stdev`	0.001
`k0`	0
`k1`	1
`k2`	6
`num_iter`	150

Table 4.8: Factorization Machine

Parameter	**Optimal Value**
`base_estimator`	Decision Stump
`learning_rate`	0.5
`n_estimators`	250

Table 4.9: AdaBoost

Parameter	**Optimal Value**
`base_estimator`	Decision Tree
`n_estimators`	1000
`max_samples`	0.5
`max_features`	0.875
`bootstrap`	False
`bootstrap_features`	False

Table 4.10: Bagging

Parameter	**Optimal Value**		
	FM	**AdaBoost**	**Bagging**
`vectorizer`	Count	Count	Count
`pre_active`	False	False	False
`stop_words`	None	None	None
`binary`	False	True	True
`ngram_range`	[1, 1]	[1, 2]	[1, 1]
`max_features`	None	None	None
`min_df`	5	50	10
`max_df`	0.9	0.7	1.0
Accuracy	**62.2%**	**61.8%**	**60.4%**

Table 4.11: Pipeline Parameters for Each Classifier

Optimal Values Found by the Grid Searches for Each Classifier

Chapter 5

Evaluation

This chapter describes the evaluation and the metrics used to analyze the prediction quality of *SentTwi*. In the first part, different metrics to measure the quality performance of a classification system are introduced. The second section of this chapter presents the actual prediction quality of *SentTwi* as well as the impact of selected components on the prediction quality.

5.1 Evaluation Metric

To visualize the performance of a classifier, a table called *confusion matrix* is used. A confusion matrix C is of size $n \times n$ with n as the number of possible classes for the particular classification problem. Table 5.1 shows a confusion matrix for our three-point sentiment classification task. Each column represents the number of tweets the system predicted for a given class, whereas each row contains the number of tweets in the actual class. Each cell $c_{i,j}$ represents the number of tweets which belong to class i and are predicted by the classifier as class j. The confusion matrix of a perfect classifier would therefore only contain large values in the diagonal cells $c_{i,i}$, which are the *correct predictions*, and zeros in the off-diagonal elements, the *incorrect predictions*.

All of the following four evaluation metrics can be easily comprehended with the help of the confusion matrix.

Accuracy An intuitive method to measure the quality of a classifier is to count the number of correct classifications. This metric is called *accuracy* and represents the ratio of correct classifications. In terms of the confusion matrix, the accuracy is the sum of true predictions along the diagonal in proportion to the number of total test samples, as

Actual Class		**Predicted Class**		
		POSITIVE	NEUTRAL	NEGATIVE
	POSITIVE	*true positive*	false neutral	false negative
	NEUTRAL	false positive	*true neutral*	false negative
	NEGATIVE	false positive	false neutral	*true negative*

Table 5.1: Confusion Matrix C for the Three Classes POSITIVE, NEUTRAL and NEGATIVE. Entries in *Italic* Represent the Correct Predictions.

presented in the following equation:

$$Accuracy = \frac{\sum_{i=1}^{n} c_{i,i}}{\sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j}} \tag{5.1}$$

Accuracy assumes equal cost for all classes since it takes every class equally in account. So if the classes are not equally distributed and one class is predominant, a classifier can achieve a high accuracy by simply predicting the predominant class. For example, if 90 of 100 samples are labeled as class A, a classifier scores an accuracy of 90% just by predicting every test item as class A.

Precision *Precision* measures the class agreement of a system by calculating the fraction of predictions which are classified correctly. In multiclass classification, precision is calculated for each class individually. Equation 5.1 defines the precision score for a given class i.

$$Precision_i = \frac{c_{i,i}}{\sum_{j=1}^{n} c_{j,i}} \tag{5.2}$$

Recall *Recall*, also referred as *sensitivity*, estimates the effectiveness of a classifier to predict a given class. This metric is defined by the fraction of correct predictions of class i and is given by the following equation:

$$Recall_i = \frac{c_{i,i}}{\sum_{j=1}^{n} c_{i,j}} \tag{5.3}$$

F-Score The *F-score* (also known as F_1*-score* or *F-measure*) is a combination of precision and recall. It is the harmonic mean of precision and recall. The F_1-score for class c_i is defined according to the following equation:

$$F_1^i = \frac{2 \cdot precision_i \cdot recall_i}{precision_i + recall_i} \tag{5.4}$$

Model	Acc (%)	F_1^{PN} (%)
FM	**64.14**	53.68
AdaBoost	63.01	**57.00**
Bagging	62.72	56.87

Table 5.2: Accuracy and F_1^{PN} of *SentTwi* Evaluated with the Official Tweet 2016 Test Dataset.

The metric used to evaluate and compare the performance of *SentTwi* throughout this thesis is a variation of the *macroaveraged F_1-score*. This F_1^{PN}-score is used by the SemEval challenge [NRR+16] and is the average of the F_1-score of the positive class F_1^P and the F_1-score of the negative class F_1^P. Equation 5.1 illustrates the calculation of the F_1^{PN}-score.

$$F_1^{PN} = \frac{F_1^P + F_1^N}{2} \tag{5.5}$$

5.2 Results

In this section, we present the evaluation results of the prediction quality of *SentTwi* for each of the three classification algorithms: factorization machine, AdaBoost ensemble and Bagging ensemble. The parameters used for the following results have been determined by the grid searches described in Section 4.6. The statistics of the used train and test datasets are given in Table 2.5: The training procedure is done on the *Total Train* dataset consisting of 22,700 different tweets. The official *SemEval 2016 Test* set (16,758 tweets) is used for evaluation.

5.2.1 Classifier Comparison

Table 5.2 illustrates the performance of *SentTwi* achieved with each of the three classifiers.
It can be seen that all three classifiers perform similarly well. Especially the two ensemble methods, AdaBoost and Bagging, achieve roughly equal accuracy and F_1-scores. Although using a factorization machine as the classification algorithm scored the best accuracy (64.14%), both ensemble methods outperform the factorization machine approach in terms of F_1^{PN} by more than 3%.

		Predicted Class			
		POSITIVE	NEUTRAL	NEGATIVE	TOTAL
Actual Class	POSITIVE	*3,510*	2,210	68	5,788
	NEUTRAL	1,556	*6,382*	520	8,458
	NEGATIVE	100	1,556	*856*	2,512
	TOTAL	5,166	10,148	1,444	16,758

Table 5.3: FM: Confusion Matrix

	positive	**neutral**	**negative**
Precision	67.94%	62.89%	59.28%
Recall	60.64%	75.46%	34.08%
F_1	64.08%	68.60%	43.28%

Table 5.4: FM: Additional Performance Results

		Predicted Class			
		POSITIVE	NEUTRAL	NEGATIVE	TOTAL
Actual Class	POSITIVE	*3,469*	2,049	270	5,788
	NEUTRAL	1,603	*5,731*	1,124	8,458
	NEGATIVE	264	889	*1,359*	2,512
	TOTAL	5,336	8,669	2,753	16,758

Table 5.5: AdaBoost: Confusion Matrix

	positive	**neutral**	**negative**
Precision	65.01%	66.11%	49.36%
Recall	59.93%	67.76%	54.10%
F_1	62.37%	66.92%	51.62%

Table 5.6: AdaBoost: Additional Performance Results

		Predicted Class			
		POSITIVE	NEUTRAL	NEGATIVE	TOTAL
Actual Class	POSITIVE	*3,687*	1,829	272	5,788
	NEUTRAL	2,003	*5,535*	920	8,458
	NEGATIVE	375	844	*1,293*	2,512
	TOTAL	6,065	8,208	2,485	16,758

Table 5.7: Bagging: Confusion Matrix

	positive	**neutral**	**negative**
Precision	60.79%	67.43%	52.03%
Recall	63.70%	65.44%	51.47%
F_1	62.21%	66.42%	51.75%

Table 5.8: Bagging: Additional Performance Results

Confusion Matrices and Additional Performance Results for Each Classifier

Table 5.3 to 5.8 show the confusion matrices as well as recall, precision and F_1-scores per class for each of the three classifiers. Looking at the confusion matrices, it is noticeable that the factorization machine approach classifies more tweets as neutral compared to the ensemble methods (10,148 vs. 8,669 and 8,208, respectively) which results in fewer correct true negative classifications (856 vs. 1,359 and 1,293, respectively). The same characteristic can be seen in the recall score of the negative class of the factorization machine classifier ($R_N = \frac{856}{2{,}512} = 34.08\%$) shown in Table 5.4. Since the F_1^N-metric is the harmonic mean of precision and recall ($F_1^N = \frac{2*0.5928*0.3408}{0.5928+0.3404} = 0.4328$), the low number of correct negative classifications is the cause for the low F_1^{PN}-score of *SentTwi* with FMs compared to the AdaBoost and Bagging classifiers.

5.2.2 Feature Analysis

To measure the impact of each type of feature, we perform a feature analysis presented in the feature ablation table 5.9. By removing every feature from the full feature set, we can compare the effectiveness of a specific feature with other features and how much it contributes to the overall performance. For this feature analysis the same training and test datasets as for the classifier comparison are used. The presented values are the averaged results of three executions to smooth the impact of randomness in the sampling process of the classification algorithms. The measured difference in the performance scores between the three executions ranges from 0.01% to 0.84%. The F_1^{PN}-scores of the FM executions fluctuate most (0.3% on average), whereas the other values deviate 0.13% on average. The reported baseline is the performance result of a baseline classifier that classifies each tweet in the test set as positive.
It can be seen that the features with the highest impact on the prediction quality are the sentiment lexicons. They improve the accuracy of all three classifiers by 1.7% to 2.79%. Interesting is the remarkable increase in the F_1^{PN}-score: 12.64% for factorization machines, 13.43% for AdaBoost and 17.93% for Bagging. These results can only be achieved if all lexicons are used while each lexicon individually has minimal impact on the performance. The second most substantial feature is the bag-of-words model. It increases the accuracy roughly the same as the sentiment lexicons (1.33% to 2.32%) but without the same effect on the F_1^{PN}-metric.

Feature	**FM**		**AdaBoost**		**Bagging**	
	Acc (%)	F_1^{PN} (%)	Acc (%)	F_1^{PN} (%)	Acc (%)	F_1^{PN} (%)
All	64.13	53.59	62.98	56.95	62.68	56.82
- Bag-of-Words	61.81 (-2.32)	49.74 (-3.84)	61.50 (-1.48)	56.36 (-0.59)	61.36 (-1.33)	55.94 (-0.88)
- POS	64.07 (-0.06)	53.58 (-0.01)	62.66 (-0.31)	56.35 (-0.60)	62.55 (-0.13)	56.61 (-0.21)
- Punctuation	63.70 (-0.44)	53.38 (-0.21)	62.55 (-0.43)	56.78 (-0.18)	62.45 (-0.23)	56.64 (-0.18)
- Emoticons	63.85 (-0.29)	53.28 (-0.31)	62.95 (-0.03)	57.01 (0.05)	62.64 (-0.04)	56.80 (-0.03)
- Sentiment Lexicons	61.34 (-2.79)	40.94 (-12.64)	61.28 (-1.70)	43.52 (-13.43)	60.01 (-2.67)	38.89 (-17.93)
- AFINN	63.86 (-0.27)	53.24 (-0.34)	62.86 (-0.11)	56.86 (-0.09)	62.54 (-0.14)	56.67 (-0.16)
- MaxDiff	63.87 (-0.26)	53.28 (-0.31)	62.80 (-0.18)	56.82 (-0.14)	62.45 (-0.24)	57.04 (0.22)
- Opinion Lexicon	63.97 (-0.17)	53.27 (-0.31)	62.80 (-0.18)	56.81 (-0.15)	62.55 (-0.14)	56.66 (-0.17)
- Sentiment140	63.84 (-0.30)	53.35 (-0.24)	62.88 (-0.10)	56.66 (-0.29)	62.48 (-0.20)	56.49 (-0.33)
- SentiStrength	63.77 (-0.36)	52.96 (-0.63)	62.99 (0.02)	56.88 (-0.07)	62.50 (-0.18)	56.58 (-0.25)
- SentiWordNet	63.90 (-0.23)	53.52 (-0.07)	62.96 (-0.02)	56.90 (-0.05)	62.61 (-0.07)	56.71 (-0.12)
- VADER	63.38 (-0.76)	51.28 (-2.31)	62.29 (-0.69)	54.51 (-2.44)	62.05 (-0.63)	54.98 (-1.84)
Baseline	34.21 (-29.92)	25.49 (-28.10)	34.21 (-28.77)	25.49 (-31.46)	34.21 (-28.47)	25.49 (-31.33)

Table 5.9: Feature Ablation for Each Classifier. Each Row Stands for the Full Feature Pipeline with the Particular Feature Removed.

Chapter 6

Discussion

The results show that FMs, AdaBoost as well as Bagging classifiers are suitable for sentiment detection of tweets. All three classification algorithms provide a comparable prediction quality. *SentTwi* with AdaBoost and Bagging achieves a similar accuracy and F_1^{PN}-score, while using a FM results in a slightly better accuracy but the lowest F_1^{PN}-score.
Since the provided results are based on the SemEval 2016 test set, we can compare our results with other SemEval participants. The results of the official SemEval-2016 Task 4 Subtask A [NRR$^+$16] are given in Appendix A.3. In terms of the F_1^{PN}-score, *SentTwi* with AdaBoost (57%) would be among the top 20 teams of 2016. Comparing our most accurate model with these teams, *SentTwi* using a factorization machine (64.14%) would achieve the third place.
As specified in the confusion matrices and performance results, the main reason for *SentTwi*'s moderate F_1^{PN}-score are the low precision and recall scores for the negative class. Our first assumption was that negations are not identified correctly and using bigrams would mitigate this issue. After further analysis of the data obtained by the grid searches, this was not the case as the results of the best pipeline settings with unigrams are very close to the scores of the best settings which utilize bigrams ($\pm$0.4% accuracy at most). In addition, both ensemble methods use different ngrams, whereas their results are similar. This observation is consistent with Go et al. [GBH09] and Pang et al. [PLV02].
It was unexpected to see that all three Feature Vector Grid Searches recommend skipping the preprocessing step for the three classification types (setting `pre_active` to `false`) before the POS tagging process gets executed. Similar to the aforementioned bigram results, *SentTwi*'s best scores with and without the preprocessing steps are very close together (less than 0.2% difference in accuracy). The reason for this outcome is unclear as various studies propose similar preprocessing processes (e.g., lowercasing, substituting or replacing phrases) [AXV$^+$11, KTM11, Ill15].

Chapter 7

Conclusion and Future Work

The main contribution of this thesis is to show that factorization machines and ensemble methods can be used to predict the sentiment of tweets. Therefore, we proposed *SentTwi*, a supervised machine learning system which uses one of three possible classification algorithms to classify the sentiment of tweets into positive, neutral or negative. Furthermore, an introduction into sentiment analysis and fundamentals of factorization machines and ensemble methods are given.
SentTwi can utilize factorization machines, AdaBoost or Bagging ensembles as the classification component of choice. By combining a variety of techniques and components to extract features from tweets, our experiments show that factorization machines and ensemble methods are useful for sentiment analysis of tweets and achieve similar results as other state-of-the-art sentiment classification systems. Features from a range of existing sentiment lexicons turn out to be the most useful set of features. The second highest performance gain is achieved with the features extracted by the bag-of-words vectorizer using either term frequency weighting for the factorization machine or term presence weighting for the AdaBoost and Bagging classifier.

Future work can improve some features of *SentTwi* to enhance the prediction quality of *SentTwi* or add new components to adapt *SentTwi* for new requirements and future trends:

Improve Negative Predictions The highest performance increase can probably be obtained by improving the prediction quality of the negative class. Better negation detection can reduce the number of wrong classifications by incorporating concepts like negation aware sentiment lexicons or negation shifters.

Five-Point Scale Currently all training and test tweets are categorized according to a three-point schema. Many real-world applications use a five-point scale to rate products, services and more: e.g., Amazon, MovieLens, and Google Play. Applying such a five-point scale would turn the existing classification problem into an ordinal regression problem (i.e., an intermediate between classification and regression). Therefore, the classifiers have to be either adapted to handle ordering or replaced with regressors.

Other Classifiers In addition to the three currently used classifiers, *SentTwi* could utilize further classification models: A random forest classifier as another ensemble method can be used to analyze the impact of randomness on sentiment classification. Another possible approach for future work would be the use of deep learning and neural network models in particular, which are becoming increasingly popular in research.

Appendix A

Appendix

A.1 ARK POS Tagger tagset

Tag	**Description**	**Examples**
N	common noun	football cars
O	pronoun	you this
ˆ	proper noun	USA Google
S	nominal + possessive	heaven's house's
Z	proper noun + possessive	Ellen's Amazon's
L	nominal + verbal	who's it's
M	proper noun + verbal	Mark'll Potter's
V	verb incl. copula, auxiliaries	join may gonna
A	adjective	awesome coooool
R	adverb	just only
!	interjection	tgif omg oh
D	determiner	her the this
P	pre- or postposition, or subordinating subordinating conjunction	for in with
&	coordinating conjunction	or & but
T	verb particle	up out
X	existential there, predeterminers	all there
Y	X + verbal	there's
#	hashtag	#ibm #news
@	at-mention	@cisco @drewbrees
˜	discourse marker, indications of continuation across multiple tweets	... :
U	URL or email address	http://t.co/123
E	emoticon	:) <3
$	numeral	one 2nd 6
,	punctuation	. , ?
G	other abbreviations, foreign words, possessive endings, symbols, garbage	smh lkaj –>

Table A.1: Tagset by Gimpel et al. [GSO+11] Used by the ARK Tagger in the POS Tagging Component.

A.2 Emoticons

Positive Emoticons

(:	*)	*-)	/o/	8)	8-)	8D	8-D	:)	:))
:*	:-)	:-))	:-*	:-B	:-D	:-P	:-]	:-b	:-p
:-}	:3	:>	:B	:D	:P	:]	:^)	:^*	:^B
:^D	:^P	:^b	:^p	:b	:c)	:o)	:o]	:o}	:p
:}	:-,	:')	:']	:'}	;)	;-)	;D	;]	;^)
;}	;-)	;-]	<3	=)	=*	=3	=B	=D	=P
=]	=^)	=^B	=^D	=^P	=^]	=^b	=^p	=^}	=b
=p	=}	=-3	=-D	=')	=']	='}	B^D	XD	X-D
\m/	\o/	\o\	^-^	^.^	^^	^_^	xD	xoxo	xx
x-D									

Neutral Emoticons

-.-	-.-'	-_-	-_-'	:#	:-#	:-@	:-x	:-|	:O
:@	:L	:S	:^#	:^@	:^o	:^x	:o	:x	:|
:-.	:-/	:-|	=#	=@	=L	=O	=X	=|	>.<
>:/	>:\	><	>_<						

Negative Emoticons

)-:	):	:$:-{	:'(	:'-(	:(	:-(	:-[	:-||
:/	:<	:[	:\	:^(	:^[	:^{	:c	:o(
:o[	:o{	:{	:-<	:-[	:-c	:'(	:'[	:'{
</3	=$	=(	=/	=[	=\	=^(	=^[	=^{
={	='(	='[	='{	>:(	>:-(	>:-=(	>:-[	>:-{
>:[	>=(	>=[	>=^(	>=^[	>=^{	>={	D-:	D8
D:	D:<	D;	D=	DX	D^:	D^=	D-':	O_o
Oo	o.O	v.v						

Table A.2: List of 91 Positive, 34 Neutral and 66 Negative Emoticons used by the Emoticon Feature Component.

A.3 SemEval-2016 Results

	F_1^{PN}		Acc	
Team	**Rank**	**%**	**Rank**	**%**
SwissCheese	1	63.3	1	64.6
SENSEI-LIF	2	63.0	7	61.7
UNIMELB	3	61.7	8	61.6
INESC-ID	4	61.0	10	60.0
aueb.twitter.sentiment	5	60.5	6	62.9
SentiSys	6	59.8	9	60.9
I2RNTU	7	59.6	12	59.3
INSIGHT-1	8	59.3	5	63.5
TwiSE	9	58.6	24	52.8
ECNU	10	58.5	16	57.1
NTNUSentEval	11	58.3	2	64.3
MDSENT	12	58.0	20	54.5
CUFE	12	58.0	4	63.7
THUIR	14	57.6	11	59.6
PUT	14	57.6	14	58.4
LYS	16	57.5	13	58.5
IIP	17	57.4	23	53.7
UniPI	18	57.1	3	63.9
DIEGOLab16	19	55.4	19	54.9
GTI	20	53.9	26	51.8
OPAL	21	50.5	22	54.1
DSIC-ELIRF	22	50.2	27	51.3
UofL	23	49.9	15	57.2
ELiRF	23	49.9	21	54.3
ISTI-CNR	25	49.4	17	56.7
SteM	26	47.8	31	45.2
Tweester	27	45.5	25	52.3
Minions	28	41.5	18	55.6
Aicyber	29	40.2	28	50.6
mib	30	40.1	29	48.0
VCU-TSA	31	37.2	32	38.2
SentimentalITists	32	33.9	29	48.0
WR	33	33.0	34	29.8
CICBUAPnlp	34	30.3	33	37.4

Table A.3: Official Results for *Subtask A: Message Polarity Classification* of the *SemEval-2016 Task 4: Sentiment Analysis in Twitter*. [NRR+16]

Bibliography

[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, 2011, pages 30–38, URL `http://dl.acm.org/citation.cfm?id=2021109.2021114`.

[BES10] S. Baccianella, A. Esuli and F. Sebastiani: *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.*, *LREC*, volume 10, 2010, pages 2200–2204.

[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, 2010, pages 36–44, URL `http://dl.acm.org/citation.cfm?id=1944566.1944571`.

[Bre96] L. Breiman: *Bagging Predictors*, Mach. Learn., volume 24(2), (1996), pages 123–140, URL `http://dx.doi.org/10.1023/A:1018054314350`.

[Car79] J. G. Carbonell: *Subjective Understanding: Computer Models of Belief Systems.*, Ph.D. thesis, New Haven, CT, USA, 1979.

[CDCX14] C. Chen, W. Dongxing, H. Chunyan and Y. Xiaojie: *Exploiting Social Media for Stock Market Prediction with Factorization Machine*, *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02*, WI-IAT '14, IEEE Computer Society, Washington, DC, USA, 2014, pages 142–149, URL `http://dx.doi.org/10.1109/WI-IAT.2014.91`.

[CHV99] O. Chapelle, P. Haffner and V. N. Vapnik: *Support Vector Machines for Histogram-Based Image Classification*, IEEE transactions on Neural Networks, volume 10(5), (1999), pages 1055–1064.

[CNM06] R. Caruana and A. Niculescu-Mizil: *An Empirical Comparison of Supervised Learning Algorithms*, *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pages 161–168.

[CS93] P. K. Chan and S. J. Stolfo: *Toward Parallel and Distributed Learning by Meta-learning*, *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases*, AAAIWS'93, AAAI Press, 1993, pages 227–240, URL `http://dl.acm.org/citation.cfm?id=3000767.3000789`.

[CS97] P. K. Chan and S. J. Stolfo: *On the Accuracy of Meta-learning for Scalable Data Mining*, Journal of Intelligent Information Systems, volume 8, (1997), pages 5–28.

[DB95] T. G. Dietterich and G. Bakiri: *Solving Multiclass Learning Problems via Error-correcting Output Codes*, J. Artif. Int. Res., volume 2(1), (1995), pages 263–286, URL `http://dl.acm.org/citation.cfm?id=1622826.1622834`.

[DC01] S. R. Das and M. Y. Chen: *Yahoo! for Amazon: Sentiment parsing from Small Talk on the Web*, *8th Asia Pacific Finance Association Annual Conference*, 2001.

[Die00a] T. G. Dietterich: *Ensemble Methods in Machine Learning*, *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, Springer-Verlag, London, UK, UK, 2000, pages 1–15, URL `http://dl.acm.org/citation.cfm?id=648054.743935`.

[Die00b] T. G. Dietterich: *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization*, Mach. Learn., volume 40(2), (2000), pages 139–157, URL `http://dx.doi.org/10.1023/A:1007607513941`.

[DLP03] K. Dave, S. Lawrence and D. M. Pennock: *Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews*, *Proceedings of the 12th in-*

ternational conference on World Wide Web, ACM, 2003, pages 519–528.

[DLY08] X. Ding, B. Liu and P. S. Yu: *A Holistic Lexicon-Based Approach to Opinion Mining*, *Proceedings of the 2008 international conference on web search and data mining*, ACM, 2008, pages 231–240.

[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, 2013.

[ES06] A. Esuli and F. Sebastiani: *SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining*, *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC06*, 2006, pages 417–422.

[FCD$^+$00] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer and D. Haussler: *Support vector machine classification and validation of cancer tissue samples using microarray expression data*, Bioinformatics, volume 16(10), (2000), pages 906–914.

[FS96] Y. Freund and R. E. Schapire: *Experiments with a New Boosting Algorithm*, *Proceedings of the 13th International Conference on Machine Learning*, volume 96, 1996, pages 148–156.

[FSTR11] C. Freudenthaler, L. Schmidt-Thieme and S. Rendle: *Bayesian Factorization Machines.*

[GBH09] A. Go, R. Bhayani and L. Huang: *Twitter Sentiment Classification using Distant Supervision*, CS224N Project Report, Stanford, volume 1(2009), (2009), page 12.

[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, 2011, pages 42–47, URL http://dl.acm.org/citation.cfm?id=2002736.2002747.

[HDD13] L. Hong, A. S. Doumith and B. D. Davison: *Co-Factorization Machines: Modeling User Interests and Predicting Individual Decisions in Twitter*, *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, ACM, New York, NY, USA, 2013, pages 557–566, URL http://doi.acm.org/10.1145/2433396.2433467.

[HG14] C. Hutto and E. Gilbert: *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*, *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*, 2014.

[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, 2004, pages 168–177, URL http://doi.acm.org/10.1145/1014052.1014073.

[HLY$^+$12] Z. Han, X. Li, M. Yang, H. Qi, S. Li and T. Zhao: *HIT at TREC 2012 Microblog Track.*, *TREC*, volume 12, 2012, page 19.

[Ho98] T. K. Ho: *The Random Subspace Method for Constructing Decision Forests*, IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 20(8), (1998), pages 832–844, URL http://dx.doi.org/10.1109/34.709601.

[HS90] L. K. Hansen and P. Salamon: *Neural Network Ensembles*, IEEE Trans. Pattern Anal. Mach. Intell., volume 12(10), (1990), pages 993–1001, URL http://dx.doi.org/10.1109/34.58871.

[HZZT00] F. J. Huang, Z. Zhou, H.-J. Zhang and C. Tsuhan: *Pose Invariant Face Recognition*, Institute of Electrical and Electronics Engineers, Inc., 2000, URL https://www.microsoft.com/en-us/research/publication/pose-invariant-face-recognition/.

[Ill15] M. Illecker: *Real-time Twitter Sentiment Classification based on Apache Storm*, Master's thesis, University of Innsbruck, 2015.

[JED04] W. Jiang, G. Er and Q. Dai: *Multiple Boosting SVM Active Learning for Image Retrieval*, *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 3, IEEE, 2004, pages 421–424.

[KABO10] A. Karatzoglou, X. Amatriain, L. Baltrunas and N. Oliver: *Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering*, *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, ACM, New York, NY, USA, 2010, pages 79–86, URL `http://doi.acm.org/10.1145/1864708.1864727`.

[Kor09] Y. Koren: *The BellKor Solution to the Netflix Grand Prize*, Netflix prize documentation, volume 81, (2009), pages 1–10.

[KTM11] E. Kouloumpis, W. Theresa and J. D. Moore: *Twitter Sentiment Analysis: The Good the Bad and the OMG!*, *Proceedings of the Fifth International Conference on Weblogs and Social Media*, AAAI Press, 2011, pages 538–541.

[KV94] A. Krogh and J. Vedelsby: *Neural Network Ensembles, Cross Validation and Active Learning*, *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, MIT Press, Cambridge, MA, USA, 1994, pages 231–238, URL `http://dl.acm.org/citation.cfm?id=2998687.2998716`.

[KZM] S. Kiritchenko, X. Zhu and S. M. Mohammad: *Sentiment Analysis of Short Informal Texts*, volume 50, , pages 723–762.

[Lew98] D. D. Lewis: *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval*, *European conference on machine learning*, Springer, 1998, pages 4–15.

[LFM15] J. J. Louviere, T. N. Flynn and A. Marley: *Best-worst scaling: Theory, Methods and Applications*, Cambridge University Press, 2015.

[LG12] G. Louppe and P. Geurts: *Ensembles on Random Patches*, P. A. Flach, T. De Bie and N. Cristianini (eds.), *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pages 346–361, URL `http://dx.doi.org/10.1007/978-3-642-33460-3_28`.

[LHC05] B. Liu, M. Hu and J. Cheng: *Opinion Observer: Analyzing and Comparing Opinions on the Web*, *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, ACM, New York, NY, USA, 2005, pages 342–351, URL `http://doi.acm.org/10.1145/1060745.1060797`.

[LIRK03] L. V. Lita, A. Ittycheriah, S. Roukos and N. Kambhatla: *Truecasing*, *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, Association for Computational Linguistics, 2003, pages 152–159.

[Liu12] B. Liu: *Sentiment Analysis and Opinion Mining*, Synthesis lectures on human language technologies, volume 5(1), (2012), pages 1–167.

[LL08] H.-T. Lin and L. Li: *Support Vector Machinery for Infinite Ensemble Learning*, J. Mach. Learn. Res., volume 9, (2008), pages 285–312, URL `http://dl.acm.org/citation.cfm?id=1390681.1390690`.

[MCGVF+13] E. Martınez-Cámara, Y. Gutiérrez-Vázquez, J. Fernández, A. Montejo-Ráez and R. Munoz-Guillena: *Ensemble classifier for Twitter Sentiment Analysis.*

[MKZ13] S. M. 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, 2013.

[MMS93] M. P. Marcus, M. A. Marcinkiewicz and B. Santorini: *Building a Large Annotated Corpus of English: The Penn Treebank*, Comput. Linguist., volume 19(2), (1993), pages 313–330, URL `http://dl.acm.org/citation.cfm?id=972470.972475`.

[Moh11] S. Mohammad: *Colourful Language: Measuring Word-colour Associations*, *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, CMCL '11, Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, pages 97–106, URL `http://dl.acm.org/citation.cfm?id=2021096.2021107`.

[MRS08] C. D. Manning, P. Raghavan and H. Schütze: *Introduction to Information Retrieval*, Cambridge University Press, New York, NY, USA, 2008.

[Nie11] 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`.

[NMPS+09] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, W. X. Shang and Y. F. Zhu: *Winning the KDD Cup Orange Challenge with Ensemble Selection*, *Proceedings of the 2009 International Conference on KDD-Cup 2009 - Volume 7*, KDD-CUP'09, JMLR.org, 2009, pages 23–34, URL `http://dl.acm.org/citation.cfm?id=3000364.3000366`.

[NRR+16] P. Nakov, A. Ritter, S. Rosenthal, V. Stoyanov and F. Sebastiani: *SemEval-2016 Task 4: Sentiment Analysis in Twitter*, *Proceedings of the 10th International Workshop on Semantic Evaluation*, SemEval '16, Association for Computational Linguistics, San Diego, California, 2016.

[NY03] T. Nasukawa and J. Yi: *Sentiment Analysis: Capturing Favorability Using Natural Language Processing*, *Proceedings of the 2Nd International Conference on Knowledge Capture*, K-CAP '03, ACM, New York, NY, USA, 2003, pages 70–77, URL `http://doi.acm.org/10.1145/945645.945658`.

[OLL+14] R. J. Oentaryo, E.-P. Lim, J.-W. Low, D. Lo and M. Finegold: *Predicting Response in Mobile Advertising with Hierarchical Importance-Aware Factorization Machine*, *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, ACM, New York, NY, USA, 2014, pages 123–132, URL `http://doi.acm.org/10.1145/2556195.2556240`.

[OM99] D. Opitz and R. Maclin: *Popular Ensemble Methods: An Empirical Study*, Journal of Artificial Intelligence Research, volume 11, (1999), pages 169–198.

[OOD^{+}13] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider and N. A. Smith: *Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters*, *In Proceedings of NAACL 2013*, Association for Computational Linguistics, 2013.

[PL08] B. Pang and L. Lee: *Opinion Mining and Sentiment Analysis*, Foundations and trends in information retrieval, volume 2(1-2), (2008), pages 1–135.

[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*, Association for Computational Linguistics, 2002, pages 79–86.

[Pol90] J. B. Pollack: *Backpropagation is Sensitive to Initial Conditions*, volume 4, 1990, pages 269–280.

[POL10] S. Petrović, M. Osborne and V. Lavrenko: *The Edinburgh Twitter Corpus*, *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics in a World of Social Media*, WSA '10, Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pages 25–26, URL `http://dl.acm.org/citation.cfm?id=1860667.1860680`.

[Por01] M. F. Porter: *Snowball: A language for stemming algorithms*, Website, 2001, URL `http://snowballstem.org/texts/introduction.html`, accessed on 22 Januray, 2017.

[PP10] A. Pak and P. Paroubek: *Twitter as a Corpus for Sentiment Analysis and Opinion Mining*, *Proceedings of the International Conference on Language Resources and Evaluation*, volume 10 of *LREC 2010*, 2010, pages 1320–1326.

[PT10] G. Paltoglou and M. Thelwall: *A Study of Information Retrieval Weighting Schemes for Sentiment Analysis*, *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pages 1386–1395, URL `http://dl.acm.org/citation.cfm?id=1858681.1858822`.

[QLY13] R. Qiang, F. Liang and J. Yang: *Exploiting Ranking Factorization Machines for Microblog Retrieval*, *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, ACM, New York, NY, USA, 2013, pages 1783–1788, URL `http://doi.acm.org/10.1145/2505515.2505648`.

[Ren10] S. Rendle: *Factorization Machines*, *2010 IEEE International Conference on Data Mining*, IEEE, 2010, pages 995–1000.

[Ren12a] S. Rendle: *Factorization Machines with libFM*, ACM Trans. Intell. Syst. Technol., volume 3(3), (2012), pages 57:1–57:22, URL `http://doi.acm.org/10.1145/2168752.2168771`.

[Ren12b] S. Rendle: *Learning Recommender Systems with Adaptive Regularization*, *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, ACM, New York, NY, USA, 2012, pages 133–142, URL `http://doi.acm.org/10.1145/2124295.2124313`.

[RGFST11] S. Rendle, Z. Gantner, C. Freudenthaler and L. Schmidt-Thieme: *Fast Context-aware Recommendations with Factorization Machines*, *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, ACM, New York, NY, USA, 2011, pages 635–644, URL `http://doi.acm.org/10.1145/2009916.2010002`.

[RNK^{+}15] S. Rosenthal, P. Nakov, S. Kiritchenko, S. Mohammad, A. Ritter and V. Stoyanov: *SemEval-2015 Task 10: Sentiment Analysis in Twitter*, *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, Association for Computational Linguistics, Denver, Colorado, 2015, pages 451–463, URL `http://www.aclweb.org/anthology/S15-2078`.

[Rok05] L. Rokach: *Ensemble Methods for Classifiers*, Springer US, Boston, MA, 2005, pages 957–980, URL `http://dx.doi.org/10.1007/0-387-25465-X_45`.

[Sch90] R. E. Schapire: *The Strength of Weak Learnability*, Machine Learning, volume 5(2), (1990), pages 197–227, URL `http://dx.doi.org/10.1007/BF00116037`.

[SF01] A. K. Seewald and J. Fürnkranz: *An Evaluation of Grading Classifiers*, *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, IDA '01, Springer-Verlag, London, UK, UK, 2001, pages 115–124, URL `http://dl.acm.org/citation.cfm?id=647967.741617`.

[SS00] R. E. Schapire and Y. Singer: *BoosTexter: A Boosting-based System for Text Categorization*, Machine learning, volume 39(2-3), (2000), pages 135–168.

[TBP$^+$10] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai and A. Kappas: *Sentiment in Short Strength Detection Informal Text*, Journal of the American Society for Information Science and Technology, volume 61(12), (2010), pages 2544–2558.

[TBP12] M. Thelwall, K. Buckley and G. Paltoglou: *Sentiment Strength Detection for the Social Web*, Journal of the American Society for Information Science and Technology, volume 63(1), (2012), pages 163–173.

[TKMS03] K. Toutanova, D. Klein, C. D. Manning and Y. Singer: *Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network*, *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, Association for Computational Linguistics, Stroudsburg, PA, USA, 2003, pages 173–180, URL `http://dx.doi.org/10.3115/1073445.1073478`.

[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 linguistics*, Association for Computational Linguistics, 2002, pages 417–424.

[VC14] G. Vinodhini and R. Chandrasekaran: *Opinion mining using principal component analysis based ensemble model*, CSI Transactions on ICT, volume 2(3), (2014), pages 169–179.

[Wan08] X. Wan: *Using Bilingual Knowledge and Ensemble Techniques for Unsupervised Chinese Sentiment Analysis*, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pages 553–561, URL `http://dl.acm.org/citation.cfm?id=1613715.1613783`.

[WB83] Y. Wilks and J. Bien: *Beliefs, Points of View, and Multiple Environments*, Cognitive Science, volume 7(2), (1983), pages 95–119.

[WBR$^+$10] M. Wiegand, A. Balahur, B. Roth, D. Klakow and A. Montoyo: *A Survey on the Role of Negation in Sentiment Analysis*, *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, NeSp-NLP '10, Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pages 60–68, URL `http://dl.acm.org/citation.cfm?id=1858959.1858970`.

[Wol92] D. H. Wolpert: *Stacked Generalization*, Neural networks, volume 5(2), (1992), pages 241–259.

[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, 2005, pages 347–354, URL `http://dx.doi.org/10.3115/1220575.1220619`.

[ZIS16] E. Zangerle, M. Illecker and G. Specht: *SentiStorm: Realtime Sentiment Detection of Tweets*, HMD Praxis der Wirtschaftsinformatik, volume 53(4), (2016), pages 514–529, URL `http://dx.doi.org/10.1365/s40702-016-0237-6`.

[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 and Dublin City University, Dublin, Ireland, 2014, pages 443–447, URL `http://www.aclweb.org/anthology/S14-2077`.