NTNU – Trondheim
Norwegian University of
Science and Technology

Classification and Visualisation of Twitter Sentiment Data

Mikael Brevik
Øyvind Selmer

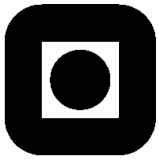
Master of Science in Computer Science
Submission date: May 2013
Supervisor: Bjørn Gamback, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Classification and Visualisation of Twitter Sentiment Data

Mikael Brevik & Øyvind Selmer

Master's Thesis



Department of Computer and Information Science
Norwegian University of Science and Technology

May 30, 2013

Abstract

The social micro-blog site Twitter grows in user base each day and has become an attractive platform for companies, politicians, marketeers, and others wishing to share information and/or opinions. With a growing user market for Twitter, more and more systems and research are released for taking advantage of its informal nature and doing opinion mining and sentiment analysis.

This master thesis describes a system for doing Sentiment Analysis on Twitter data and experiments with grid searches on various combinations of machine learning algorithms, features and preprocessing methods to achieve so. The classification system is fairly domain independent and performs better than baseline.

This system is designed to be fast enough to classify big amounts of data and tweets in a stream, and provides an application program interface (API) to easily transfer data to applications or end users.

Three visualisation applications are implemented, showing how to use the API and providing examples of how sentiment data can be used.

The main contributions are:

C1 A literary study of the state-of-the-art for Twitter Sentiment Analysis.

C2 The implementation of a general system architecture for doing Twitter Sentiment Analysis.

C3 A comparison of different machine learning algorithms for the task of identifying sentiments in short messages in a fairly semi-independent domain.

C4 Implementations of a set of visualisation applications, showing how to use data from the generic system and providing examples of how to present sentiment analysis data.

Sammendrag

Den sosiale mikrobloggingsnettstedet Twitter er i kontinuerlig vekst og har blitt et attraktivt verktøy for bedrifter, politikere, markedsførere og andre som ønsker å dele informasjon og/eller meninger. Med økende brukermasse, kommer det ut flere og flere systemer og forskningsresultater relatert til Twitter, sentimentanalyse og meningsmålinger.

Denne masteroppgaven definerer et system for å utføre sentimentanalyse på data fra Twitter og oppnår dette ved å eksperimentere med parametersøk på forskjellige kombinasjoner av maskinlæringsalgoritmer, funksjoner og preprosesseringsmetoder. Det utviklede systemet er forholdsvis domeneuavhengig og har bedre ytelse enn baseline.

Dette systemet ble designet for å takle store mengder data og klassifisere tweets i datastrømmer. For å enkelt tilby data til sluttbrukeren, ble et "application program interface" (API) definert.

Et sett med visualiseringsapplikasjoner ble implementert for å vise bruken av API-et og hvordan sentiment data kan bli brukt og presentert.

Hovedbidragene fra oppgaven er følgende:

C1 Oppgaven definerer en state-of-the-art for Twitter Sentimentanalyse (TSA).

C2 Et generelt systemarkitektur for TSA ble utviklet og implementert.

C3 Forskjellige maskinlæringsalgoritmer er satt i mot hverandre for å best finne sentiment i korte meldinger på en relativt domeneuavhengig måte.

C4 Visualiseringsapplikasjoner er implementert, og viser hvordan en kan bruke dataen fra det generelle systemet og hvordan sentimenter kan bli presentert og brukt.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfilment of the requirements for the degree of Master of Science in Computer Science.

The Master's Thesis work has been performed at the Department of Computer and Information Science, NTNU, Trondheim, with supervisor Björn Gambäck and co-supervisor Lars Bungum.

Acknowledgements

We would like to thank our supervisor Björn Gambäck and co-supervisor Lars Bungum, for invaluable response and guidance for this project and to Amitava Das for inital discussions regarding sentiment analysis and guidance during the pre-project.

For helping us with performing a literature review, we would like to thank Anders Kofod-Petersen.

We would also like to thank the organizers of SemEval'13 for the data sets used to train our system, in addition to our own little data set.

A big thanks to the human annotators, volunteering to help us generate a data set of our own.

Contents

List of Tables

List of Figures

Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
bagging	Bootstrap Aggregating
CSS	Cascading Style Sheets
DOM	Document Object Model
ESC	Eurovision Song Contest
HTTP	Hypertext Transfer (or Transport) Protocol
IDF	Inverse Document Frequency
JS	JavaScript
JSON	JavaScript Object Notation
MaxEnt	Maximum Entropy
MLE	Maximum Likelihood Estimation
MV*	Model-View-*
MVC	Model View Controller
NB	Naïve Bayes
NTNU	Norwegian University of Science and Technology

POS	Part-of-Speech
REST	Representational state transfer
RQ	Research Question
RT	Re-Tweet
SA	Sentiment Analysis
sklearn	Scikit-learn python module
SLR	Systematic Literature Review
SLRP	Systematic Literature Review Protocol
SMS	Short Message (or Messaging) Service
SVM	Support Vector Machine
TDH	Twitter Data Handler
TSA	Twitter Sentiment Analysis

Chapter 1

Introduction

The introduction chapter first defines the task description. Background and Motivation for this task is given in section 1.2, followed by an overview of the project goals in section 1.3 and in section 1.4 the project contribution is summarised. During the project, we participated in a workshop, which is described in section 1.5. In the last section of the introduction, an overview of this report is described.

This report is the result of a Master thesis at Department of Computer and Information Science, NTNU, 2013.

1.1 Task Description

The task was given by Björn Gambäck and Lars Bungum at IDI, NTNU

Sentiment Analysis using the Twitter Corpus

In recent years, micro-blogging has become prevalent, and the Twitter API allows users to collect a corpus from

their micro-blogosphere. The posts, named tweets are limited to 140 characters, and are often used to express positive or negative emotions to a person or product.

In this project, the goal is to use the Twitter corpus to do Sentiment Analysis and develop tools for visualising the results. Pak and Paroubek (2010) have shown how to do this using frameworks like Support Vector Machines (SVMs) and Conditional Random Fields (CRFs), benchmarked with a Naïve Bayes Classifier baseline. They were unable to beat the baseline, and the goal of this project will be to experiment with these and other machine learning frameworks as Maximum Entropy learners to try to beat the baseline.

1.2 Background and Motivation

Twitter has become a popular social media service often referred to as a micro-blogging site. On Twitter users can post messages of maximum 140 characters, called tweets, on their own *timeline*. A timeline is a collection of all user submitted tweets and all tweets from the other users that a user is connected to (following). Tweets can be categorized by using hashtags. A hashtag can, for instance, look like *#happy* or *#obama2012*. By annotating the tweets with this tag, users can find similar tweets across Twitter. Tweets often also contain references to other users by using the @-character followed by the user name (e.g., @*obama*) or references to pictures or articles via URLs.

Twitter has grown very rapidly and the usage statistics is ever changing. In June 2012, there were posted over 400 million tweets every day [All Twitter, 2012]. With over 500 million users, where about 170 million of these are active ones [WebProNews, 2012], it is safe to say that Twitter offers a lot of data.

The growth in Twitter users and status updates (tweets) over the last years

has made Twitter an attractive platform for companies, marketeers, politicians and others who are looking for feedback. Manually collecting information like this is a tedious if not impossible task.

The informal nature of Twitter leads to a lot of sentiments being posted and this has made Twitter a gold mine for sentiment analysis (SA). Many systems have used Twitter as corpus for SA. The first one to really use Twitter as a corpus was Sentiment140 (previously known as TwitterSentiment) by a group of Stanford students [Go et al., 2009]. After this paper, and as Twitter grew in popularity, many other systems have been developed. Some of the later ones are TwiSent and C-Feel-IT [Mukherjee et al., 2012], Tweenator [Saif et al., 2012] and MSAS [Chamlertwat et al., 2012].

Many systems are using a single step, three way, classification for sentiments. Another approach is to do a two step classification, where the subjectivity is classified first, and in step two, the polarity is predicted. *Subjectivity classification* is the task of classifying a tweet as subjective or objective. Pak and Paroubek [2010] counted word frequencies in a subjective vs an objective set of tweets; the results showed that interjections and personal pronouns are the strongest indicators of subjectivity. If the tweet is classified as objective (or neutral), no further classification is required. If the tweet is subjective, however, it requires polarity classification. *Polarity classification* will classify between positive and negative tweets. Kouloumpis et al. [2011] experimented with different solutions for tweet polarity classification, and found that the best performance came from using n-grams together with a lexicon and micro-blog features. Interestingly, performance dropped when a part-of-speech (POS) tagger was included. They speculate that this can be due to the accuracy of the POS tagger itself, or that POS tagging just is less effective for analysing tweet polarity. Gimpel et al. [2010] showed that POS taggers designed specifically for tweets can increase the accuracy.

The informal texts on social media represent challenges for traditional natural language processing systems. These texts are short, and often contain misspellings, slang and abbreviations. The challenge of handling

such a vocabulary has only been researched over the last few years.

Another interesting feature is that Twitter messages offer a lot of meta data and information about their origin, such as location, language, and more. These data could, for example, be used to filter out and classify tweets from a certain event, like a festival or a conference.

For visualising sentiment data, not as much has been done. Sentiment140, by Gimpel et al. [2011], has some rudimentary visualising with pie and bar charts.[1] Kamvar and Harris [2011] developed an emotional search engine with different advanced techniques for visualising mood, but not for three-way classification for sentiment on Twitter.

1.3 Project Goals

In this section, the main goals of this project are described.

G1 Experiment with different models for doing sentiment analysis
Design and implement different models for doing sentiment analysis. Experiment with these models and find the model with highest accuracy and beating the baseline the most.

G2 Develop tools for visualising sentiment classified tweets
Data is wasted if it is not used for anything. Data needs proper, usable, summarising and visualisation tools to be of use. One of the goals of this project is to come up with, plan and develop tools that are useful for showing the real value of sentiment classified tweets. To be able to make visualisation applications, we also need a way to distribute the data through a documented interface.

[1] `http://www.sentiment140.com/`

1.4 Contributions

C1 A literary study of the state-of-the-art for Twitter Sentiment Analysis.

C2 The implementation of a general system architecture for doing Twitter Sentiment Analysis.

C3 A comparison of different machine learning algorithms for the task of identifying sentiments in short messages in a fairly semi-independent domain.

C4 Implementations of a set of visualisation applications, showing how to use data from the generic system and providing examples of how to present sentiment analysis data.

1.5 SemEval'13

During the course of this project, a workshop for semantic analysis systems, called SemEval'13, was held. The workshop had several different shared tasks, amongst others a task for building a message polarity classification system. In relation to this workshop, data sets were published to the participants. To take advantage of this and to formally test our system, we participated in the workshop and submitted a paper. This paper was accepted for publication in the proceedings of the SemEval'13 conference [Selmer et al., 2013].

1.6 Thesis Structure

In Chapter 2, the existing solutions and current state-of-the-art are described. In Chapter 3, data sets and machine learning theory are described. The system architecture and model is presented and documented

in Chapter 4. Chapter 4 also has a complete description of the visualisation application architecture and how the applications work. The experimental setup is covered in Chapter 5 and the results in Chapter 6. Chapter 7 includes evaluation and discussions of the results. In Chapter 8 the report is concluded and future work is proposed.

Chapter 2

State-of-the-Art

A systematic literature review (SLR) was conducted to establish the state-of-the-art of a Twitter Sentiment Analysis (TSA) system. The method and results is documented in this chapter. The first section, Section 2.1, covers the introduction and defines our research questions. The review method is described in Section 2.2 and the results in Section 2.3.

As a workshop on TSA was conducted after defining the state-of-the-art, a lot of additional work has been done in the field. Updates to the state-of-the-art can be found in Section 2.4. Related work in visualisation is described in Section 2.5.

2.1 Introduction

SLRs are still new in the field of Computer Science. There are few examples of an SLR in practice. The method we used is heavily inspired by the documentation paper by Kofod-Petersen [2012] and the Master's thesis by Lillegraven and Wolden [2010].

We defined our research questions (RQ) as the following:

RQ1 What are some of the existing solutions for SA (sentiment analysis) in the Twitter Corpus?

RQ2 How does the different solutions found by addressing RQ1 compare to each other with respect to micro-blogs like Twitter?

RQ3 What is the strength of the evidence in support of the different solutions?

RQ4 What implications will these findings have when creating the application/system?

2.2 Review Method

In this section we will describe our process step by step according to the systematic literature review protocol as defined by and shown in Appendix A.

2.2.1 Planning

1. Identification of the need for a review
Assumed that a review is needed for this project.

2. Commissioning a review
Assumed to be commissioned for this project, so no commission report was produced.

3. Specifying the research question(s), RQs
We defined the RQs based on what we felt needed to be researched when finding the state-of-the-art for sentiment analysis systems on Twitter. The RQs can be seen in Section 2.1.

4. Developing a review protocol

The systematic literature review protocol SLRP was developed in the early stages of the project. After the first revision, it was evaluated by the project supervisor. The protocol was revised several times during the execution of the review.

5. Evaluating the review protocol

The protocol was evaluated by the project supervisor.

2.2.2 Conducting

1. Identification of research

We defined a series of keywords and synonyms to construct a search string to use. The search string was defined to find papers relevant to our research questions. The development of this search string is documented by Appendix A. The search string we used was:

```
("Sentiment Analysis" OR "Sentiment
Classification" OR "Opinion Mining")
AND (Twitter OR Microblog)
```

For the search domain, we used Google Scholar. Google Scholar accumulates results from several different sources and gave many results for our search. A lot of the results given corresponded to the studies handed out by the project supervisor, a domain expert.

We limited the search to only give papers released after 2008. This is due to Twitter being as new as it is.

The search resulted in 1060 papers, but after the first 8 pages of results (with 10 papers per page), we found that the papers were mostly irrelevant and we had limited resources and time for handling all 1060 papers. This resulted in a set of 80 papers, ready to go through the selection process.

2. Selection of primary studies

First, we defined a set of primary inclusion criteria. These criteria were applied to the title and abstract of the study. If a paper did not pass the criteria, it was dropped from the set. In addition we defined secondary inclusion criteria. These criteria were checked against the full text of the study. These inclusion criteria are documented in the protocol in Appendix A.

After both selection processes, we had a set of 23 papers.

3. Study quality assessment

Using the description of Kofod-Petersen [2012] we defined a set of 10 quality criteria. All of the criteria are documented in the protocol in Appendix A. Each of the papers in our set was assessed according to all of these criteria. If the paper met the criteria, it would get 1 point; if it partially met them, it would get 0,5 points; and if it did not meet the criteria, it would get 0 points.

The papers with the lowest score did not get taken as much into consideration when defining the state-of-the-art.

4. Data extraction and monitoring

We defined a set of fields and information categories we thought were important in order to answer our research questions. These data features were used to generate a table of information. The information was retrieved by reading the papers and manually extracting the data we needed.

5. Data synthesis

This step was included in the data extraction step.

2.2.3 Reporting

1. Specifying dissemination strategy

As this is for a master thesis, the result of the SLR is presented in this report.

2. Formatting the main report
The SLR was written as a section in this project report.

3. Evaluating the report
It is mandatory for an expert to review this report as well as a project supervisor, as it is a report for a master thesis.

2.3 Results

In this section the result of the systematic literature review is presented. In Section 2.3.1 all of the extracted data from the selected studies are presented. The assessed quality of the articles is also documented here.

Section 2.3.2 answers the research questions given as a part of the SLRP.

2.3.1 Selected studies

All of the selected studies are presented in table format as a part of the data extraction step in the SLR. The results can be found in Table 2.1. In addition to the data features defined in the review protocol, the total quality is added to be a part of the extraction table.

Table 2.1: Data extraction step. Showing data as per defined in the SLRP in Appendix A

ID	Authors	Title	Pub. Year	System name	ML Algorithm	Dataset	Findings & Conclusions	QA
S1	Hassan Saif, Yulan He & Harith Alani	Semantic Smoothing for Twitter Sentiment Analysis	2011	-	NB	http://twittersentiment.appspot.com/	Slight improvement by .3%	7.5
S2	Subhabrata Mukherjee, Akshat Malu, A.R. Balamurali, Pushpak Bhattacharyya	TwiSent: A Multistage System for Analyzing Sentiment in Twitter	2012	TwiSent	NB	C-Feel-IT dataset.	Improvements with filtering/ normalization. Best: 66,69% with manually annotated dataset	8.5
S3	Adam Bermingham & Alan Smeaton	Classifying Sentiment in Microblogs:Is Brevity an Advantage?	2010	-	NB, SVM	CLARITY dataset (removed due to Twitter terms), Pang & Lee's movie corpus,TREC Blogs06 corpus and a collection of microreviews from Blippr	Accuracy of 74.85% using NBC and unigrams. Easier to classify microblogs than long texts	9.0
S4	Wilas Chamlertwat, Pattarasinee Bhattarakosol, Tippakorn Rungkasiri, Choochart Haruechaiyasak	Discovering Consumer Insight from Twitter via Sentiment Analysis	2012	MSAS	SVM, Lexical (non ML)	Self compiled, manually annotated 600 tweets	Sentiment Analysis can be useful for consumer research. No accuracy for classification.	8.0
S5	Dmitry Davidov, Oren Tsur & Ari Rappoport	Enhanced Sentiment Learning Using Twitter Hashtags and Smileys	2010	-	HFW/ CW	Dataset by Brendan O?Connor. Hashtags and smileys as training labels.	Hashtags and smilies works good for SA	6.5
S6	Murphy Choy, Michelle L.F. Cheong, Ma Nang Laik & Koo Ping Shung	A sentiment analysis of Singapore Presidential Election 2011 using Twitter data with census correction	2011				Given proper recalibration using census information, the twitter information can translate into pretty accurate information about the political landscape.	5.5

Continuing next page

Continued from previous page

ID	Authors	Title	Pub. Year	System name	ML Algorithm	Dataset	Findings & Conclusions	QA
S7	Hassan Saif, Yulan He & Harith Alani	Semantic Sentiment Analysis of Twitter	2012	Tweentor	NB	Stanford Twitter Sentiment Corpus, Health Care Reform, Obama-McCain Debate	Improvements by using semantic features on wide range topics. Acc: 83.9%	9.0
S8	Lei Zhang, Riddhiman Ghosh, Mohamed Dekhil, Meichun Hsu & Bing Liu	Combining Lexicon-based and Learning-based Methods for Twitter Sentiment Analysis	2011		SVM, Lexical		Outperforms approaches that use lexical or supervised methods alone	8.5
S9	Alec Go, Lei Huang, Richa Bhayani	Twitter Sentiment Analysis	2009	Sentiment140	NB, SVM	http://www.stanford.edu/alecmgo/cs224n/twitterdata.2009.05.25.c.zip	85% - bias accuracy.	9.0
S10	James Spencer & Gulden Uchyigit	Sentimentor: Sentiment Analysis of Twitter Data	2012	Sentimentor	NB		52% for three classes(pos,neg and objective) using bigrams without POS tagging	7.5
S11	Amir Asiaee T, Arindam Banerjee, Mariano Tepper & Guillermo Sapiro	If You are Happy and You Know It... Tweet	2012		NB, SVM, k-NN, DL		82.95% accuracy with NB on tweets about the weather	8.5
S12	Finn Ärup Nielsen	A new ANEW: Evaluation of a word list for sentiment analysis in microblogs	2011		Lexical	Labeled language data created with Amazon Mechanical Turk(AMT)	The AFINN word list performs slightly better than ANEW in Twitter sentiment analysis	7.5
S13	Luciano Barbosa & Junlan Feng	Robust sentiment detection on Twitter from biased and noisy data	2010	TwitterSA	SVM	Used Twendz, Twitter Sentiment and TweetFeel to collect data	By using data with noisy labels as input, they achieved a more abstract representation of Twitter messages than raw words.	8.5

Continuing next page

Continued from previous page

ID	Authors	Title	Pub. Year	System name	ML Algorithm	Dataset	Findings & Conclusions	QA
S14	Younggue Bae & Hongchul Lee	A Sentiment Analysis of Audiences on Twitter: Who Is the Positive or Negative Audience of Popular Twitterers?	2011		Used LIWC-2007 dictionary to extract sentiment	Collected tweets from celebrities and their mentions		5.5
S15	Mauro Cohen, Pablo Damiani, Sebastian Durandeu, Renzo Navas, Hernán Merlino, Enrique Fernández	Sentiment Analysis in Microblogging: A Practical Implementation	2011	-	NB	Manually gathered and annotated. 1500 tweets	Not as good accuracy as previous systems	4.0
S16	Roberto González-Ibáñez, Smaranda Muresan, Nina Wacholder	Identifying Sarcasm in Twitter: A Closer Look	2011	-	SMO, LogR	Data collected by using hashtag search. 900 tweets	Best result (75.89%) was achieved in the polarity-based classification P-N. Automatic classification can be as good as human classification; however, the accuracy is still low	9.5
S17	Akshi Kumar, Teeja Mary Sebastian	Sentiment Analysis on Twitter	2012	-	None. POS and own alg.		The initial results show that it is a motivating technique. No stated accuracy	7.5
S18	Alexander Pak, Patrick Paroubek	Twitter as a Corpus for Sentiment Analysis and Opinion Mining	2010	-	NB	http://www.stanford.edu/alecmgo/cs224n/twitterdata.2009.05.25.c.zip	63% accuracy using bigrams and POS tagging	10
S19	Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, Tiejun Zhao	Target-dependent Twitter Sentiment Classification	2011	-	SVM	Subjectivity: Manually annotated 727 tweets for each class. Polarity: Manually annotated 268 tweets for each (pos, neg)	Subjectivity: 85.6%. Polarity: 68.2%	8.5

Continuing next page

Continued from previous page								
ID	**Authors**	**Title**	**Pub. Year**	**System name**	**ML Algorithm**	**Dataset**	**Findings & Conclusions**	**QA**
S20	Kun-Lin Liu, Wu-Jun Li & Minyi Guo	Emoticon Smoothed Language Models for Twitter Sentiment Analysis	2012	ESLAM	Unigram, MLE	Sanders Corpus (5513 manually labeled tweets with one of the four different topics: Apple, Google, Microsoft, and Twitter)	ESLAM performs better than both emoteicons(distant supervised) and manually annotated tweets(fully supervised) alone	10
S21	Efthymios Kouloumpis, Theresa Wilson, Johanna Moore	Twitter Sentiment Analysis: The Good the Bad and the OMG!	2011	-	AdaBoost.MH	Gathered by hash, http://twittersentiment.blogspot.com and iSieve Corporation for evaluating data	Hash + Emoticons result in 75% accuracy	8.0
S22	Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, Rebecca Passonneau	Sentiment Analysis of Twitter Data	2011	-	SVM	Data by NextGen Invent (NGI). Manually annotated 11,875 tweets, non-bias tweets	A gain of 4% on unigram 3-way classification. Acc: 60.83%	10
S23	Asli Celikyilmaz, Dilek Hakkani-Tür & Junlan Feng	Probabilistic Model-based Sentiment Analysis Of Twitter Messages	2010		Used LDA to extract a polarity lexicon	Collected 2 million tweets using Twitter search API from September 2009 to June 2010. Keywords related to mobile operation. Made two manually annotated subsets from this collection.	Relatively 10% better F-measure with unigrams than baseline for classification with text normalization and all word unigram, bigram and trigrams as features.	7.5

In Table 2.2 all the individual points for the quality criteria are presented. The criteria are defined as a part of the review protocol. The average criterion score was 8.0.

2.3.2 Twitter Sentiment Analysis: State-of-the-Art

This section presents some of the state-of-the-art Twitter Sentiment Analysis (TSA) approaches, and the techniques that are used. The vocabulary used on Twitter makes it hard for traditional natural language processing systems to understand, because they are usually trained on a more formal language. This has made researchers exploit some of the special features that the web language, and Twitter give us, e.g., abbreviations and emoticons.

2.3.2.1 Data Collection and Preprocessing

Most of the data used in TSA research is collected through the Twitter API, either by searching for a certain topic/keyword or by streaming real-time data. Some datasets from related research on the Twitter platform has also been made available for other research projects, as an alternative to collecting a complete data set from scratch. Some approaches specialize on certain domains, while others query for tweets containing emoticons (*':)', ':)'*) to train a cross-domain classifier [Go et al., 2009]. The idea behind the emoticon approach is to make sure that the collected tweets contain subjectivity, but these training sets alone are limited to binary classification only (positive/negative classification).

After the data has been collected, it commonly goes through a filtering process. First, all non-English tweets are removed, then the Twitter specific symbols and functions described in Table 2.3 would normally be filtered out. As mentioned, a study by Go et al. [2009] used ':)' and ':(' as a label for the polarity in their training data, and thus they did not remove these emoticons, but the URLs and usernames were replaced by a nomial

#ID	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	QC9	QC10	Total
S1	1.0	0.5	1.0	0.5	1.0	1.0	0.5	0.5	0.5	1.0	7.5
S2	1.0	1.0	1.0	0.5	1.0	1.0	1.0	0.5	0.5	1.0	8.5
S3	1.0	1.0	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	9.0
S4	1.0	1.0	0.5	0.5	1.0	1.0	1.0	0.5	0.5	1.0	8.0
S5	1.0	0.5	0.5	0.5	1.0	1.0	0.0	0.5	1.0	0.5	6.5
S6	1.0	0.5	0.5	0.5	0.0	0.5	0.0	0.5	1.0	1.0	5.5
S7	1.0	1.0	1.0	0.5	0.5	1.0	1.0	1.0	1.0	1.0	9.0
S8	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.5	0.5	1.0	8.5
S9	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.5	1.0	9.0
S10	1.0	1.0	0.5	1.0	1.0	1.0	0.5	0.5	0.5	0.5	7.5
S11	1.0	0.5	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	8.5
S12	1.0	1.0	0.0	1.0	0.5	1.0	0.0	1.0	1.0	1.0	7.5
S13	1.0	1.0	1.0	1.0	0.5	0.5	0.5	1.0	1.0	1.0	8.5
S14	0.5	0.5	0.5	1.0	0.0	0.5	0.5	0.5	0.5	1.0	5.5
S15	1.0	0.0	0.0	0.5	1.0	1.0	0.0	0.0	0.0	0.5	4.0
S16	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	9.5
S17	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.5	0.0	0.5	7.5
S18	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	10,0
S19	1.0	1.0	0.5	0.5	1.0	1.0	0.5	1.0	1.0	1.0	8.5
S20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	10,0
S21	1.0	1.0	1.0	0.5	1.0	1.0	0.5	0.5	0.5	1.0	8.0
S22	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	10,0
S23	0.5	1.0	1.0	0.5	0.5	0.5	0.5	1.0	1.0	1.0	7.5
										Avg	**8.0**

Table 2.2: Quality Assessment for the studies. Each QC can give 0, 0.5 or 1 point. The average score is 8.0

RT	Retweet	Reposting another user's tweet
@	Mention	Tag used to mention another user
#	Hashtag	Hashtags are used to tag a tweet to a certain topic. Have become popular recently, and is also used on other platforms
:),:-),^^	Emoticon	Hashtags are used to tag a tweet to a certain topic. Have become popular recently, and is also used on other platforms
URL	URL	Typically a link to an external resource, e.g a new article or a photo

Table 2.3: Features that are usually removed from the tweets.

('URL' or 'USERNAME'). They also removed the query term from the text so that it would not affect the classification.

Kouloumpis et al. [2011] used a hashtagged data set (HASH) in addition to an emoticon data set (EMOT) from http://sentiment140.com. The hashtagged set is a subset of the Edinburgh Twitter corpus which consists of 97 million tweets [Petrovic et al., 2010].

Some approaches have also experimented with normalizing the tweets, and removing redundant letters, e.g "loooove" and "crazyyy", that are often used in tweets. Redundant letters like these are sometimes used to express a stronger sentiment, and it has therefore been experimented with trimming down to one additional redundant letter('loooove' = 'loove' instead of love), so that the stronger sentiment can be taken into consideration by a score/weight adjustment for that feature.

Part-of-speech tagging

Part-of-Speech (POS) tagging is a well-known process for marking the words of a sentence. Adjectives, adverbs and personal pronouns have been shown to be good indicators for subjectivity, which has made POS

tagging a good technique for filtering out objective tweets before the polarity classification. Early research on TSA showed that the challenging vocabulary made it harder to tag the tweets with a good accuracy; however, in 2010 Gimpel et al. [2011] made a POS tagger that aimed at marking tweets. It performed very well in their experiments (almost 90% accuracy).

2.3.2.2 Subjectivity Classification

The most used strategy for TSA is a two-step strategy where the first step is subjectivity classification and the second step is the polarity classification. The goal for the subjectivity classification is to separate subjective and objective tweets.

One of the most used techniques for this task is POS tagging. Pak and Paroubek [2010] found several indicators of subjectivity by counting word frequencies in a subjective set versus an objective set. They found that interjection and personal pronouns were the strongest indicators of subjectivity in their set. In their paper, Pak and Paroubek [2010] concluded that utterances were a strong indicator of subjectivity, but referring to the tag UH. According to the Wall Street Journal mark-up guidelines [Santorini, 1990], the tag UH is, however, not utterances, but rather interjections. Jiang et al. [2011] used normalization, POS tagging, word stemming, and syntactic parsing for the subjectivity classification task. The idea was that normalization of features would give better recall. According to their experimental results, their approach greatly improves the performance of sentiment classification.

Previous research has also explored the use of noisy data and distant supervised methods such as emoticons and hashtags for the subjectivity classification, where any match from a given lexicon will classify the tweet as subjective.

2.3.2.3 Polarity Classification

The final part of the analysis is the polarity classification (positive/negative). While TSA is not yet considered mature, SA for longer texts, i.e., documents and reviews, has been explored for years, see, e.g., Pang and Lee [2008] for an overview. Different techniques and algorithms that have proven worthy for longer texts have also been applied to sentence level SA with various success. Among these techniques, supervised learning methods like Naïve Bayes classification (NB), maximum entropy (MaxEnt) and support vector machines (SVM) are the most used. The limited amount of attributes in tweets makes the feature vectors shorter than in documents. For that reason there is no guarantee that algorithms that perform well on document-level SA will be the best alternatives for classifying short texts like tweets. Among the machine learning algorithms that perform well on TSA are NB, SVM and MaxEnt, see section 3.2. While the SVM technique normally beats NB and MaxEnt on longer texts, it seems to have some trouble with outperforming the NB when the feature vectors are shorter, i.e., on shorter texts. Bermingham and Smeaton [2010] have shown this in their comparison of SVM and NB for microblogs.

Some approaches have also experimented with a combination of lexicon-based methods and machine learning [Mudinas et al., 2012]. They performed an entity-level sentiment analysis as the first step. Then they used tweets that are likely to be opinionated in a lexicon-based method. The last step of their process is to train a classifier to assign the sentiment value. This approach makes it possible to train the classifier without manually labeling the data, as they are using the data from the lexicon-based method.

Kouloumpis et al. [2011] tried different solutions for the polarity classification, and found that the best performance came from using n-grams feature selection together with lexicon features and microblog features. Interestingly, the performance dropped when they included a POS tagger. They do not explain the reason for this, but speculate that it can be the ac-

curacy of the POS tagger itself, or just that POS tagging is less effective for analysing the polarity of tweets.

Supervised learning

Supervised learning methods require some sort of training data to create an inferred function for classification tasks. These data would preferably be manually annotated texts, but as this can be a labour-intensive task, some research has experimented with emoticons or a collection of hashtags as labels for positive/negative tweets. This is done by making assumptions, such as that all tweets containing positive emoticons are positive, and that all that contain negative emoticons are negative.

Unsupervised learning

The lexicon-based method seems to be the most used approach of the unsupervised methods in TSA. This technique requires a lexicon with a sentiment score for each word. When using such lexica the classifier can look up all the words in the feature vector, e.g., a bag of words, and check the sentiment score if the feature exists in the lexicon. Hence it will not need any training beforehand.

Popular sentiment lexica are SentiWordNet and the General Inquirer. Some researchers have also made custom extensions of these lexica that include manually annotated emoticons and hashtags as well as words. Nielsen [2011] made a sentiment lexicon called AFINN, specialized for Twitter. It contains a lot of words from the vocabulary used in social networks. AFINN supports slang and abbreviations, e.g., 'n00b', 'lol' and 'wtf'. This lexicon was made as a response to the ANEW lexicon which works better for document level SA since it does not support the Twitter language [Nielsen, 2011].

Another lexicon made specifically for Twitter is the NRC-Canada Lexicon, developed for the SemEval'13 shared tasks. See more information below.

2.4 State-of-the-Art Post SemEval'13

The SLR was performed before the SemEval'13 tasks [Nakov et al., 2013]. This means that a lot of work has been done on TSA after the state-of-the-art was defined. Many of the points are still relevant, but some additional information can be extracted.

In relation to SemEval'13, Mohammad et al. [2013] defined and implemented a state-of-the-art system. The system implemented by Mohammad et al. [2013], is described as the strongest system by Nakov et al. [2013], showing the highest F-measure on both tweets and SMS. Hence we will describe it in further detail below.

2.4.1 Sentiment Lexicon

The system developed by Mohammad et al. [2013], used lexica to help classify sentiment. In addition to using existing general lexica like the NRC Emotion Lexicon [Mohammad and Turney, 2010, Mohammad and Yang, 2011], the MPQA Lexicon [Wilson et al., 2005], and the Bing Liu Lexicon [Hu and Liu, 2004], Mohammad et al. [2013] also used Twitter specific lexica. One Twitter specific lexicon was developed for the system by streaming from the Twitter API from April to December 2012, using 78 different seed words as hash tags [Mohammad et al., 2013]. In addition to their own Twitter specific lexicon, the Sentiment140 lexicon created by Go et al. [2009] was used.

2.4.2 Classifier and Features

As with many previous systems (e.g., Bermingham and Smeaton [2010], Chamlertwat et al. [2012], Zhang et al. [2011], Asiaee T et al. [2012]), used Mohammad et al. [2013] supervised learning with SVM. The classification was done in one step, classifying three ways (neutral, positive

and negative).

URLs and user names were normalised to placeholder text (*http://someurl* and $@someuser$) and the tweets were tokenized and part-of-speech tagged using a tool for doing natural language processing on Twitter by Gimpel et al. [2010].

The features used were word ngrams, character ngrams, number of words in all caps, POS tags, hashtags, lexica, punctuations, emoticons, the number of elongated words (e.g., $goooood$), clusters, and the number of negated contexts. Mohammad et al. [2013] found that using all these features gave the best performance.

2.5 Visualisation: State-of-the-Art

The distribution and visualisation of sentiment data has not been given as much focus as the actual classification of the data. Sentiment140,[1] the system developed by Go et al. [2009], uses searching with bar charts and pie charts to visualise positive and negative sentiment. In addition it shows all tweets the sentiment is calculated from, even the neutral tweets.

TweetFeel[2] is also a popular site for visualisation of Twitter sentiment. It was used by Barbosa and Feng [2010] to generate a data set for their experiments. TweetFeel shows a textual stream of tweets matching a given search query. The matched query or keyword in the tweet text are highlighted as either green or red, depending on the sentiment.

Another system, using a different presentation, is SMM.[3] SMM is a Sentiment Analysis tool developed as a open source project and distributed at Github.[4] The visualisation of SMM is a bit more complex than the previ-

[1] `http://www.sentiment140.com/`
[2] `http://www.tweetfeel.com/`
[3] `http://smm.streamcrab.com/`
[4] `https://github.com/cyhex/smm`

ous systems. The user can choose whether or not to stream data, and there are five different views used to plot sentiment. Three views have plots focusing on showing negative, positive and neutral tweets developed over time, where the x-axis indicates seconds and the y-axis shows accumulated score. Another view in SMM shows a pie chart with percentage of neutral, positive and negative tweets. The last view shows the individual tweets and their sentiment score as defined by a number between -1 and 1.

Although WeFeelFine, by Kamvar and Harris [2011] is not a SA system like the previously noted systems, it is worth mentioning as it focuses a lot on visualising data partly collected from Twitter. WeFeelFine is a system that has tracked feelings from blogs and Twitter since 2005. The feelings can be anything from hunger to sleepiness. The data is presented in a creative way, where a collection of moving graphical nodes, float around on the page, represents each feeling felt. One can search and filter on feelings, gender, weather, location and date. When clicking on a node, the text that the feeling was extracted from is shown.

Chapter 3

Material and Methods

This chapter covers the material and theory for the methods used in this thesis. The first section describes the data that was used to train and evaluate the models that were developed. Most of the data was given by an ongoing workshop, SemEval'13, who hosted a shared Twitter SA task. It is also described how a smaller in-house data set was collected. The last section explains the machine learning algorithms used in the experiments.

3.1 Data sets

Manually collecting information from Twitter would be a tedious task, but Twitter offers a well documented Representational State Transfer Application Programming Interface (REST API) which allows users to collect a corpus from the microblogosphere. Most of the data used in TSA research is collected through the Twitter API, either by searching for a certain topic/keyword or by streaming realtime data. Four different data sets were used in the experiments described below. Three were supplied by the organisers of the SemEval'13 shared task on Twitter sentiment analysis [Nakov et al., 2013], in the form of a training set, a smaller initial

development set, and a larger development set. All sets consist of manually annotated tweets on a range of topics, including different products and events.

Tweet-level classification (Task 2B) was split into two subtasks in SemEval'13, one allowing training only on the data sets supplied by the organisers (constrained) and one allowing training also on external data (unconstrained). To this end, a web application[1] for manual annotation of tweets was built and used to annotate a small fourth data set ('NTNU'). Each of the 461 tweets in the 'NTNU' data set were annotated by one person only.

The distribution of target classes in the data sets is shown in Table 3.1. The data was neither preprocessed nor filtered, and thus contain hashtags, URLs, emoticons, etc. However, all the data sets provided by SemEval'13 had more than three target classes (e.g., 'objective', 'objective-OR-neutral'), so tweets that were not annotated as 'positive' or 'negative' were merged into the 'neutral' target class.

	Training		Dev 1		Dev 2		NTNU	
Class	Num	%	Num	%	Num	%	Num	%
Negative	1288	15	176	21	340	26	86	19
Neutral	4151	48	144	45	739	21	232	50
Positive	3270	37	368	35	575	54	142	31
Total	8709		688		1654		461	

Table 3.1: The data sets used in the experiments

Due to Twitter's privacy policy, the given data sets do not contain the tweet text, but only the tweet ID which in turn can be used to download the text. The Twitter API has a limit on the number of downloads per hour, so SemEval'13 provided a Python script to scrape texts from https://twitter.com. This script was slow and did not download the texts for all tweet IDs in the data sets, so a faster and more precise download

[1] `https://github.com/mikaelbr/tweetannotator`

script[2] for Node.JS was implemented and submitted to the shared task organisers.

3.2 Algorithms

The following section covers some theory for the machine learning algorithms that are used in the experiments for this thesis. First the Naïve Bayes classifier is described, and then Maximum Entropy and Support Vector Machines are covered. At the end of the section, two ensemble methods, bagging and boosting are explained.

3.2.1 Naïve Bayes Classification

The Naïve Bayes (NB) classifier is a practical Bayesian learning model that is easy to understand and implement. For some classification tasks, it has proven to be equally performing to more complex machine learning algorithms like artificial neural networks and decision trees [Mitchell, 1997]. NB is used for learning tasks where an instance x consists of a number of attribute-value pairs, and the target function $f(x)$ consists of a finite number of values from a set V.

The NB classifier is based on the assumption that all the attribute values are conditionally independent given the target value of the instance.

$$v_{NB} = \arg\max_{v_j \epsilon V} P(v_j) \prod_i p(a_i|v_j) \tag{3.1}$$

To classify an instance, the classifier uses the Maximum Likelihood Estimation (MLE) method to find the ratio of an attribute value and a given target value on the same instance in the training corpus. This means that it has to calculate the probability estimate P for each attribute a_i, given

[2] `https://github.com/mikaelbr/twitscraper`

the target value v_j, as shown in equation 3.1. It then assigns the target value as the one that gives the highest product from multiplying all the probabilities P from the training data.

The assumption of conditionally independent attributes makes this classifier best suited for handling bag-of-words models, which are composed by collections of unigrams.

3.2.2 Maximum Entropy

Maximum Entropy (MaxEnt) is a multinomial logistic regression model that allows for classification with more than two discrete classes. It has been used for various natural language tasks, such as POS tagging and text segmentation [Nigam et al., 1999].

The principle in MaxEnt is to model all that is known and assume nothing about that which is unknown. In other words, if you have some knowledge about a domain, choose a model that is consistent with the knowledge, but otherwise as uniform as possible.

The MaxEnt models are feature-based, and in binary classification scenarios it is the same as general logistic reasoning. Unlike NB it has no assumptions of conditionally independence, and can therefore be used with feature selection methods like n-grams and extended unigrams (unigrams with negation support) [Go et al., 2009].

3.2.3 Support Vector Machines

Support Vector Machines (SVM) are often called the large margin classifiers because SVM try to separate learning data with the highest possible margin [Fletcher, 2009]. By separating classes in the training data with a high margin, it will obtain a better accuracy when classifying unobserved instances. Basically, SVM can only classify binary problems, however by

dividing the problem into several subclassifications, SVM can be used for multi-class tasks also.

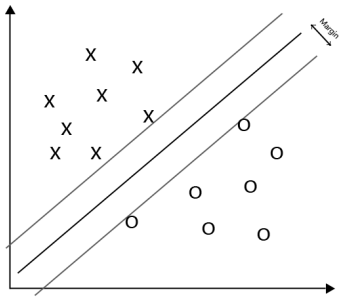


Figure 3.1: The hyperplane linearly separates the data into two classes. The points closest to the hyperplane are the support vectors.

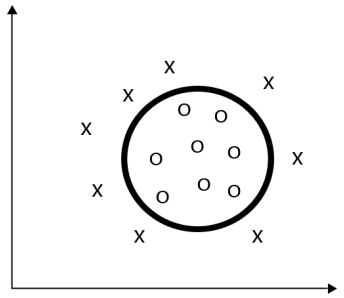


Figure 3.2: An example of a non-linear problem. A kernel function is used to map these data points into a higher dimensional feature space.

The equation for the separating hyperplane is defined by the closest points to the margin, as seen in Figure 3.1 These points are called *support vectors*. In cases where the data is non-linear, as in Figure 3.2, a kernel function is used to project the data into a high-dimensional space where it is linearly separable.

3.2.4 Ensemble Learning

Ensemble Learning is a process where multiple models are used in an ensemble to solve a problem. The thought behind using an ensemble of classifiers is to reflect how we make choices in our real lives. A normal procedure is to collect the opinion from several others, sometime even experts, to support our decisions. This is also what an Ensemble Learning classifier does. Ensemble Learning is itself a supervised learning algorithm, since it is trained and used to make predictions. An Ensemble learner can be a combination of different algorithms, and need not necessarily consist

of one specific type. There are mainly two complementary approaches to training these individual classifiers: bootstrap aggregating (bagging) and boosting. The main difference is how the classifiers in an ensemble use their training data. Each classifier in bagging is trained on a subset of the training data, while in boosting, all classifiers are trained on the same original data set [Bauer and Kohavi, 1999].

3.2.4.1 Bootstrap Aggregating

In the bagging variant, each classifier has an equally weighted vote. To obtain variance in the ensemble, the individual classifiers are trained on a different dataset, called "bootstrap replicate", that is randomly drawn from the original training data set, with replacement. Thus, for a training set with n samples, the bootstrap replicate would also have n samples, but some samples may appear several times and some samples from the original set may not appear at all.

3.2.4.2 Boosting

Boosting is a method that is used to increase the accuracy of several weak learners. Each classifier in a boosting algorithm is trained on the same dataset. Weights are added to each training instance in the learning set. The weights indicate the importance of an instance to the current classifier [Freund and Schapire, 1997].

When a classifier fails to predict the correct class for an instance, the weight for that particular instance is increased. So when training the next hypothesis, it will have an increased chance to successfully handle that instance.

In the process of generating hypotheses for the ensemble, each of the classifiers are evaluated and given a weight, as an indicator of its accuracy. When new queries are sent to the boosting algorithm, a voting takes place. Each hypothesis recommends a classification for the query instance, then

the weights are summed up and the target class with the highest total weight is chosen.

Chapter 4

Architecture

This chapter defines the architecture for the generic Twitter Sentiment Analysis system, and the implemented visualisation applications. Section 4.1 describes the architecture for the API layer as well as the sentiment classifier server. Section 4.2 documents how the visualisation applications are implemented and how the finished product works.

4.1 TSA Architecture

This section describes the overall architecture and how the system works. First the general system will be described, then the Application Programming Interface (API) Layer and classification server in turn.

To make the system as modularized and responsive as possible, the API Layer was written in Javascript, on the Node.js platform, and the sentiment classifier in the Python programming language. Both systems are continuously running servers. This allows multiple services to run simultaneously, both for the API layer and the classifier. The idea is to make the system horizontally scalable.

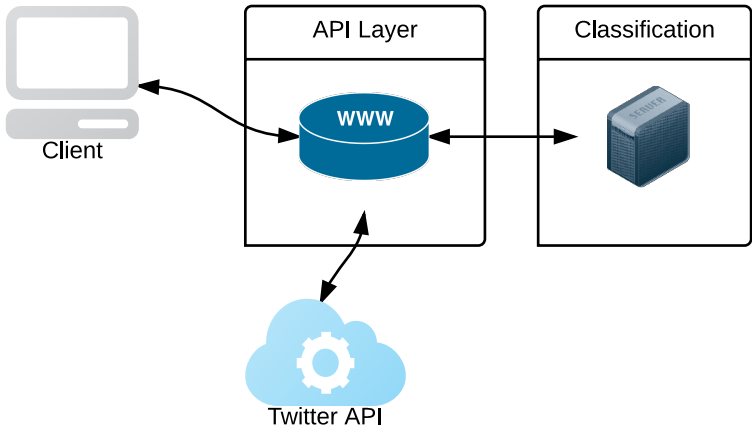


Figure 4.1: Architectural overview of the system. The client retrieves data from the Twitter API and uses the classification server for sentiment classification.

A client makes a request to the API Layer, with the same interface as the Twitter API service. From there the API Layer will retrieve information from the Twitter API with Hypertext Transfer (or Transport) Protocol (HTTP) requests, iterate over all tweets received, and send them in parallel to the classification server. When the classification server is done processing and classifying the tweet, it is sent back to the API Layer. When the API layer has received all the tweets, it responds to the client with the same JavaScript Object Notation (JSON) data structure as the Twitter API sends out, only with an additional attribute noting the tweet's sentiment. This architecture and application flow can be seen in Figure 4.1.

4.1.1 API Layer Extension

To be able to have a scalable and responsive solution, the API Layer was written using the Node.js platform. Since Node.js uses JavaScript as programming language, the JSON data retrieved from the Twitter Representational state transfer (REST) and Streaming API are easily manipulated and passed around.

The API Layer works as a thin layer extending the Twitter API. This means that the interface used by Twitter, with all defined options and appropriate methods, is reflected through the API Layer. The main benefit is that all documentation for the Twitter API also documents most of this extended API Layer.

For authentication, an application is registered with a developer account at the Twitter Developer site. This creates OAuth credentials, which are used to identify the application [Hammer-Lahav, 2010], and to gain access to the Twitter data. For this implementation, the data is retrieved using the OAuth access for the application, not at user level.

4.1.1.1 Architectural Flow

When a request from a client is made, the request gets processed by the server and the routing module determines what the client is looking for. When the proper service is found, the client-specified parameters are sent directly to the Twitter API, using the Twitter Data Handler module (TDH). The TDH module then iterates over all found tweets, and sends them in parallel to the classification server. When a tweet has been processed by the classification server, the classified sentiment is sent back to the TDH module and the original tweet object is extended to contain a property with the sentiment. When all tweets are classified, the TDH module passes the extended twitter data to the render module. The render module renders the JSON data and sends it to the client with appropriate HTTP headers set. This application flow can be seen in Figure 4.2.

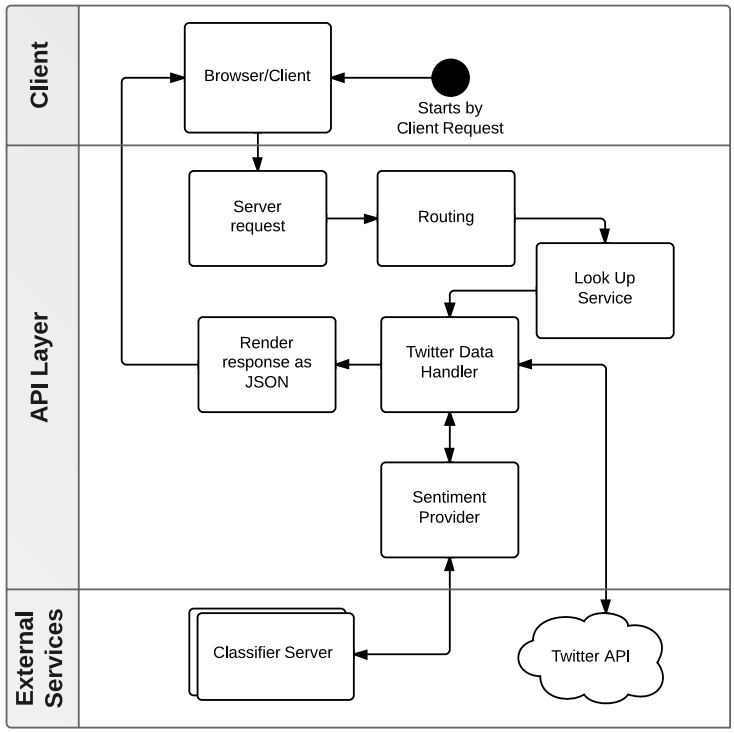


Figure 4.2: Architectural overview of the API Layer. A request is handled by the server, sending it to routing where it is processed and sent to service look-up. If a service is found, a request is sent to the Twitter API and the received data is extended by the Twitter Data Handler module to contain a sentiment. When all of the Twitter data are extended, the data is given as a response to the requesting client.

If there is an error during any part of the process, the error is caught by the routing module, and the error is rendered as a JSON object, in the same manner as it would be by the Twitter API.

When using both the Twitter REST API and Streaming API, there is a high level of asynchronism. Especially when streaming, it is impossible to predict when the next tweet is received. Due to this the system designed

needs to be able to handle this dynamic data flow. Node.js is an event-driven platform and has a natural support for asynchronous data.

All internal and external message passing in the API Layer is asynchronous. When requesting Twitter for data, an event is triggered when that data is ready and all tweets are separately sent to the classifier. By sending all tweets separately in parallel, classification of the entire set of tweets does not take much longer than classifying only one tweet.

When streaming, the TDH module opens a connection to the Twitter API, but never closes it. There is a continuously open connection to the Twitter server, which is feeding the TDH module with single tweets as they get stored in the Twitter system. From the first received tweet, a connection to the requesting client is opened by the render response module. This connection will also remain open. In this way there is an open connection between the client and the API Layer as well as between the API Layer and the Twitter API. The API Layer works as a middleman, taking in tweets, classifying them, and streaming them to the client. By running this entire process asynchronously, the system can process data independently of when it is published.

4.1.2 Sentiment Analysis Classifier

Python is computationally stronger than Node.js in many ways. Additionally, it is much more mature. There are a lot of well documented packages for handling various tasks. Scikit-learn[1] (sklearn) is one of these packages. sklearn is a package built on top of the Python packages numpy, scipy and matplotlib. sklearn integrates machine learning algorithms as SVM, NB, MaxEnt, and more. sklearn implements solutions for doing feature extraction, grid searching, cross validation, and a lot more for analysing text. Thus it is a good choice for the process of sentiment analysis. As a dynamic typed language, Python allows for rapid development

[1] `http://scikit-learn.org/`

and prototyping. These attributes are some of the reasons for why Python is a good fit for the present system.

The Sentiment Analysis Classifier system runs as a server waiting for requests. The HTTP method POST is used for a client to send a *stringified* tweet object to the server. Stringify is a JSON method for returning a serialized object represented by a string. The classification server converts this string to a Python dictionary. The response will be a string with the sentiment classification, that is, either *positive*, *neutral* or *negative*. The classification scheme can be extended if necessary.

The classifier server can be initialized with different settings for classification strategy, what port to run at, what training data to use, and whether or not to show debug data. This allows for multiple servers running at the same time, with different settings. Running multiple server instances makes it easier to compare different classification strategies. Two servers could run side by side, and a test framework could use the two servers to classify the same tweets for a comparison.

When the server is initiated, the selected model is trained and made available for the classification server.

The classification server uses a pool of child processes. For each receiving tweet, it spawns a new child from this pool and in this process the tweet is classified. This way the classification server can process several tweets in parallel, which helps the one-to-one relationship between a tweet on the classification server and the same tweet on the API Layer.

4.1.2.1 Architectural Flow

The Sentiment Provider module from the API Layer makes a request to the classifier's POST Server. The POST server translates the string to a Python dictionary and passes the information down to the Child Process Spawner. A new process is spawned and, using the module generated when initiating the server, the tweet is classified and returned to the

API Layer.

When the classification model is trained on the server initialization, various text filters, normalizations and other pre-processing methods can be utilized, as seen in figure 4.3. The model can be generated as either a one-step process, two-step process or a combination using boosting.

If a one-step model is used, one algorithm is used to classify the tweet as either negative, neutral or positive. If a two-step model is used, the tweet is first classified as either subjective or neutral in the subjectivity classification step. If it is neutral, the model returns with the classification. If the result is subjective, the tweet is sent to the polarity classification step, where the result can either be negative or positive. The end classification is returned to the API Layer.

When using the boosting model, a set of sub-models is generated and all used in conjunction to predict a sentiment of a tweet. All sub-models predicts and sends the classification to a weighted voting mechanism. The final classification is the result of the vote. This process is visualised in figure 4.4.

4.1.2.2 Classification Model Structure

The classification models are implemented by wrapping machine learning algorithms from sklearn in a inheritance-based class structure. By having every model inherit from a base model, the interface is the same across every model, and the system can use the model without having knowledge of which kind of algorithm it uses. An overview of this structure is presented in figure 4.5.

The base parent model implements methods for training the machine learning algorithm and for predicting either a set of documents or one document. For one-step algorithms as NB, SVM and MaxEnt, these generic methods can be used, but the two-step and boosting models have their own implementation.

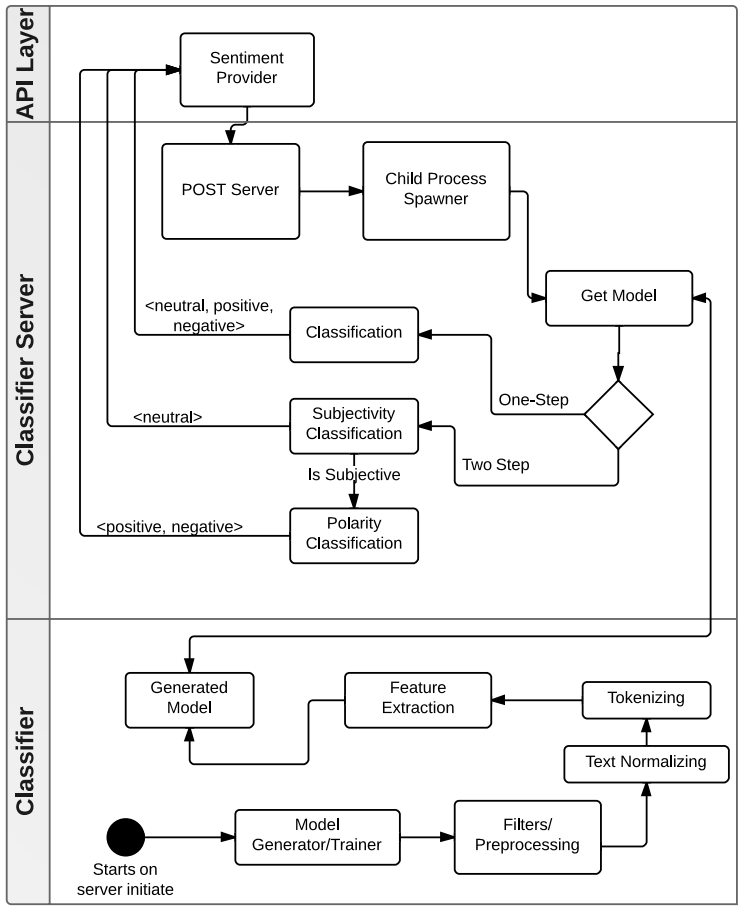


Figure 4.3: Architectural overview of the classification server. On server start, a model for predicting sentiment is generated. When a request from the API Layer is made to the POST Server, a child processes is spawned. The tweet text is extracted and sent into the model for classification. If the classification model is a one-step process, the classifier returns to the sentiment provider with either a neutral, positive or negative classification. If the generated model is two-stepped, the tweet is first classified as either neutral or subjective. If it is neutral, it is returned to the API Layer, if it is subjective, it is sent to the next step and the result from that step is returned to the sentiment provider.

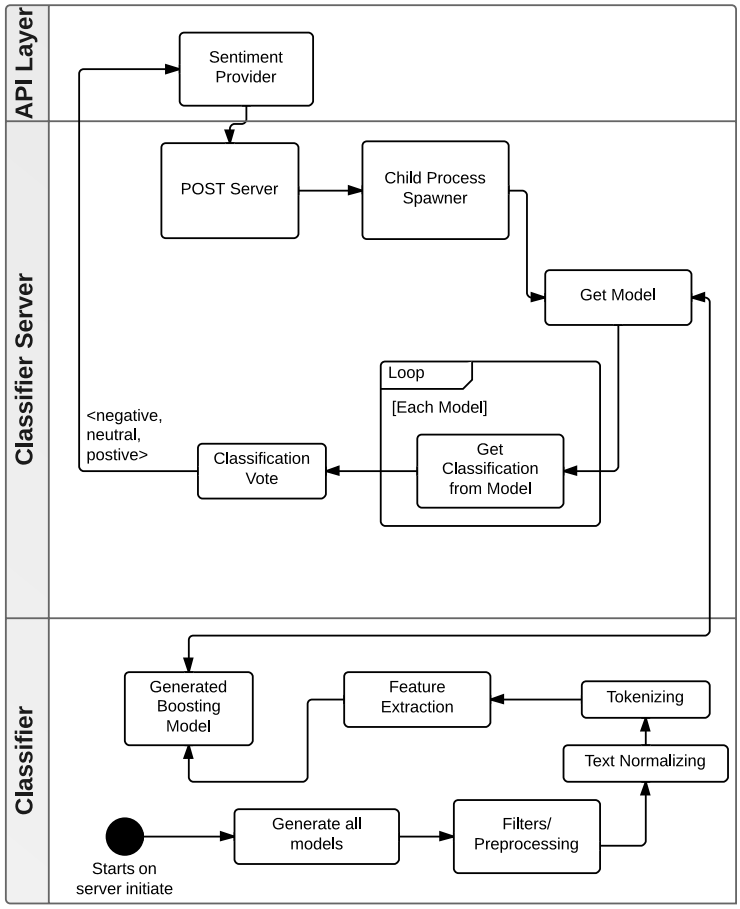


Figure 4.4: Architectural overview of the classification server with boosting. On server start, the boosting model for predicting sentiment is generated with a set of sub-models. When a request from the API Layer is made to the POST Server, a child processes is spawned. The tweet text is extracted and sent into the model for classification. Each model predicts a sentiment and sends the sentiment to voting. For voting the boosting model selects the sentiment with highest score and returns this to the API sentiment provider.

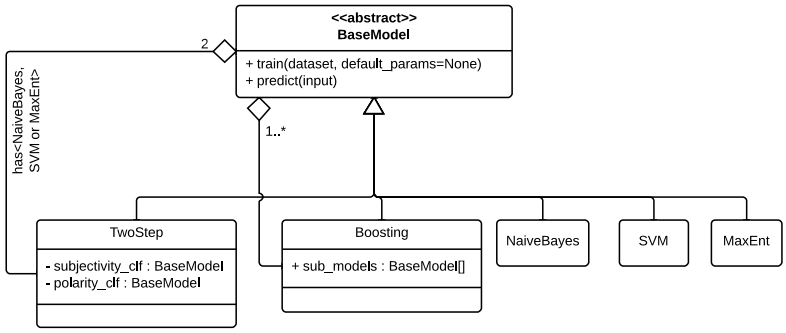


Figure 4.5: Overview of how the models are built and connected. There is a base model class implementing a method for predicting and training. All models extend from this base model. The models for Naïve Bayes, SVM and MaxEnt use the base model's implementation of train and predict, whilst the TwoStep and boosting models overrides the default one. The interface for each model is the same.

4.2 Visualisation Applications

To test and give examples of how to use the generic system, three different visualisation applications were implemented: SentiMap which presents geo-location based streams, SentiGraph which is a method for displaying tweet sentiment in a combined pie and bar chart, and SentiStack which helps visualising the twitter data as a bar stack per search query. Each of these applications is described in their own subsection. The subsections are in turn divided into describing how the applications are implemented and the finished product.

4.2.1 SentiMap: Geo-location Based Streams

To show how the streaming API and geo-location can be used to analyse sentiments for a location in real-time, SentiMap was developed. SentiMap is a visualisation tool showing the distribution of sentiments across the states of USA with or without a search query. When a new tweet is posted, within the USA, it is used to change the colour of the state it originated from. SentiMap can show the specifics of a state, with the number of positive and negative tweets, and the total of tweets registered.

SentiMap also tracks changes in sentiment in the entire USA per second. If a major event happens, the time line could show a change in sentiment across the country. The sentiment difference is calculated by taking the number of positive tweets subtracted by the number of negative tweets in a given moment.

4.2.1.1 Implementation

Most of the application is implemented on the client side, and thus running in the browser. There is a thin server side code base running, handling the stream. In this section, the server will be described first, then all details of the client implementation are covered.

Server side implementation
The client side (browser), cannot connect to a Twitter stream by itself. So to be continually fed tweets in real-time, a server implementation is necessary. SentiMap's server handles the following:

1. Connect to the API Layer stream.
2. Open a WebSocket connection for clients to connect to.
3. Start a HTTP server, serving HTML, JavaScript (JS), Cascading Style Sheets (CSS), images and other resources.
4. Serve the clients connected to the WebSocket with tweets.

Since the API layer is designed as a thin layer on top of the official Twitter API, the interface is the same. This means that programming libraries designed to interact with the Twitter API, can also communicate with the API Layer. SentiMap uses a forked (branched/copied an open sourced project) Node.JS module called nTwitter.[2] The only alteration made on nTwitter is to change the base URL for the API location from pointing to Twitter's servers and directed to the API Layer server instead.

The server opens a WebSocket connection to allow clients to stream tweets from the forked nTwitter module. WebSockets[3] is a technology used to accomplish a full-duplex connection over the Transmission Control Protocol (TCP). This means that a server can push information to a client, without the client requesting it first, as with regular HTTP connection. There are two channels of communication opened for the WebSockets. One channel for streaming all tweets from the US, and one channel used to transmit tweets related to a search. All clients share a connection to the non-query based stream, but for a search each client has its own connection. If there are no clients connected to the non-query based stream, the connection to the API Layer is closed, and only opened again if a client connects to the WebSocket.

[2] `https://github.com/AvianFlu/ntwitter`
[3] `http://en.wikipedia.org/wiki/WebSocket`

The server uses a simple Node.js HTTP module to serve static file content to clients. The static content is HTML files, JavaScript libraries and code bases, images and styling files.

When a client connects to the WebSocket, on either channel, it gets fed tweets in real time. The clients receive full tweet objects directly from the API Layer. This way the clients themselves can choose what to do with the tweets and use them for multiple purposes at once. All logic regarding computation of sentiment statistics, graphing and logging is handled by each client.

Client side implementation
The client side uses a MV* design pattern to accomplish modularity and structure. A Model-View-* (MV*) design pattern resembles a classical Model View Controller (MVC) pattern, but uses no controllers. Instead it relies more on views to handle the logic. To help with this design pattern, a JavaScript framework called Backbone.js[4] is used. In later years Backbone has become a very popular framework to use on the client side for achieving structure in large-scale applications. Backbone is a small framework and works in smaller applications as well as big corporate ones.

Every application function is its own independent module, and as such is pluggable. To support this modularity, the code is separated into three different code structures:

1. Models
2. Collections
3. Views

Models holds values and operations to interact with these values. For the SentiMap application, there are three different models; $State$, $Timeline-Stats$, and $TweetCount$.

[4] `http://backbonejs.org/`

The $TimelineStats$ model holds a two-dimensional array with time stamps and sentiment difference as value, and an operation to append attributes to this value. When a set of time stamp and sentiment difference is added to the array, the model broadcasts this change to any listeners, using *events*.

The $TweetCount$ model operates in the same way as $TimelineStats$, but instead of having an array as a value, it simply stores an integer. $TweetCount$ offers operations to increase by one or reset the integer. When the integer is changed, the model announces so through events. Both the $TimelineStats$ and $TweetCount$ models are only intended to be initiated once, unlike the $State$ model.

For each state in the USA, an object is created from the $State$ model. This model holds values for state ID (name abbreviation, e.g., NY for New York), number of positive and negative tweets for the given state. The model has operations for increasing both the negative and positive counter values by one. The model triggers an event when the values change.

A collection is simply a collection of models. In the SentiMap applications, the only collection is states which holds all the state model objects. Collections offers operations for fetching objects based on ID. This way, a collection can retrieve the model object for New York, based on the abbreviated name.

Views handles most of the logic and interactions with the end user. A view can represent a model or collection, but is not required to do so. A view is coupled with an HTML element in the Document Object Model (DOM) and can be used to render the contents of a model or collection of models into this DOM structure. A view can be looked at as a pluggable module. SentiMap consists of several views. Below is a list of the essential ones and a description of the view's role in the system.

App

The App view is the heart of the application. It loads and initiates all modules (views). When App is initiated, the map, timeline, tweet counter and stream view are initiated as well. The App view has an operation (method) for switching between a query-based stream

(search) or non-query based.

PublicStreamView

The PublicStreamView and SearchView are strongly related. PublicStreamView handles the streaming of tweets when no search query is defined. PublicStreamView connects to the non-query based channel of the server's WebSocket, and attaches an operation to respond when the server pushes a new tweet. This response operation reads the tweet and based on the tweet location property, determines what state the tweet originates from. Using the ID for the state, the state model object is fetched from the state collection. Depending on whether the sentiment classification of the tweet is positive or negative, the response operation triggers the increase negative or positive tweet method on the model.

The PublicStreamView module also has methods for disconnecting from the server WebSocket, and removing the generated HTML view from the DOM.

The response handler broadcasts a global event each time a new tweet is received.

SearchView

The SearchView works in almost exactly the same way as the PublicStreamView. If the App triggers a change between modes (from non-query based to search), the SearchView connects to the search channel on the WebSocket and communicates on which query it would like to receive tweets related to. The SearchView and PublicStreamView share the response handler on a new tweet.

MapView

To render out a map of the USA and connect each state with a view, the MapView is initiated. The MapView is the view of the states collection. When initiated, the MapView renders out a map of the USA consisting of individual parts for each state. MapView creates a StateView object per state in the map and attributes these objects with their corresponding state models.

StateView

A state view is created by the MapView, and has two attribute values; a state HTML object from the map, and the state model object.

On initialization, the StateView listens for any events on the attached model. If a state model changes its polarity (either positive or negative values are increased or reset), two operations get triggered: change colour on the map for the given state and if the state specific statistics is visible, update the statistics.

The state view also handles the presentation of the state specific statistics. If a state on the map is clicked, the StateView of that state hides the current detailed statistics (if any), and shows the statistics for the clicked state. Before showing the chart, it makes sure that the details are up to date.

TweetCountView

The TweetCountView is a simple view, initiated by the App view. On initialization, the TweetCountView attaches a view render operation on the new tweet event broadcasted by the public stream and search stream response handlers. This view renderer updates a counter on the site, informing the end user on how many tweets the SentiMap has registered on the current map.

TimelineStatsView

The TimelineStatsView is initiated from the App view, and has the $TimelineStats$ model attached to it. The TimelineStatsView renders an empty graph when it is initialized. When new sentiment difference data is pushed to the attached model, the generated graph is updated to reflect these changes. The TimelineStatsView graph is updated every second.

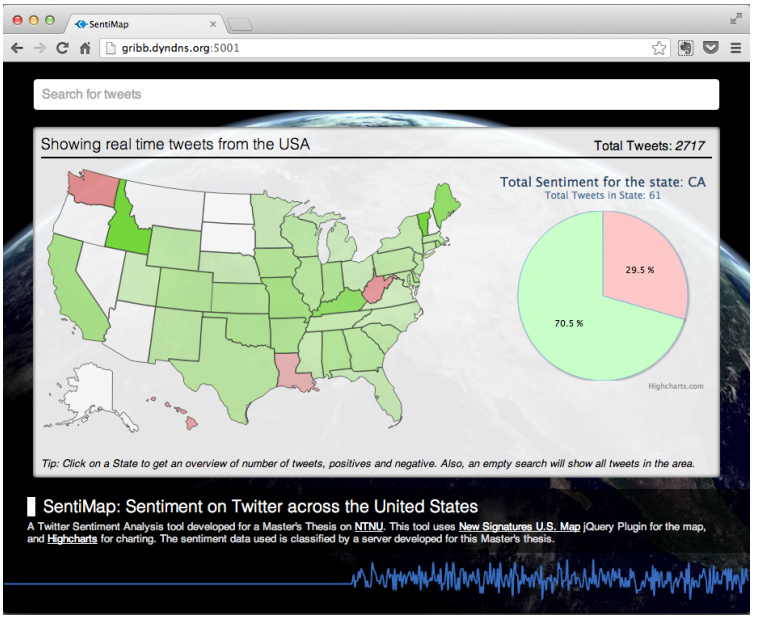


Figure 4.6: Screen shot of the SentiMap system running.

4.2.1.2 Working Product

The SentiMap applications consists of three main components: an input box for search, a map of the USA with an optional details view, and a timeline graph (see Figure 4.6). Per default, the search box is empty which results in the map being updated with tweets without any constraints regarding topic or query. If a query is typed into the search box and the return key is pressed, the map view is cleared for any colour and statistics, and the map is updated with sentiments from tweets related to the submitted query (see Figure 4.7).

When a tweet is received, the state from where the tweet has its origin updates its colour, to reflect the change in sentiment based on the classification of that given tweet. A state where there are only negative tweets has a deep red colour, and a state with only positive tweets is clear green. If there is the same amount of positive and negative tweets, the colour of

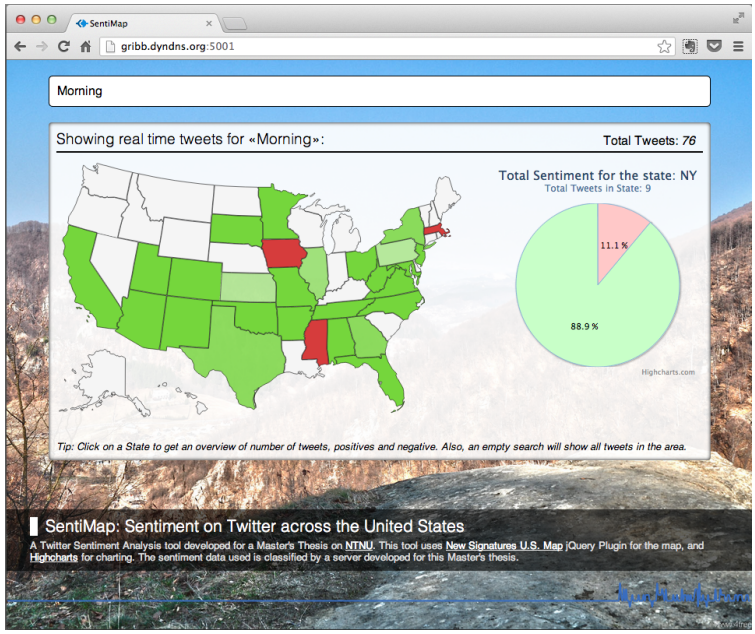


Figure 4.7: Screen shot of the SentiMap when searching for a query.

the state will be light grey. For more positive or negative a state is, the deeper is the colour of the state, i.e., if a state is 55% positive it is filled light green, and if a positive tweet from that state is registered, the state colour changes to a slightly darker green.

By default, no details from a specific state is shown. But if a user clicks on a state, a pie chart is added to the right of the U.S. map. This pie chart shows the details of sentiment on a given state, and the total of tweets registered from that state. This pie chart also updates in real time.

The timeline graph at the bottom of the page reflects the sentiment difference over time in the entire U.S. This graph is updated every second, so if on that second there is registered 18 positive tweets and 12 negative tweets, the graph will show the value 6. If there are no tweets registered for a given time, the graph shows the value 0. Hovering over a point on the x-axis of the graph will show the details of that value, as seen in Fig-

ure 4.8.

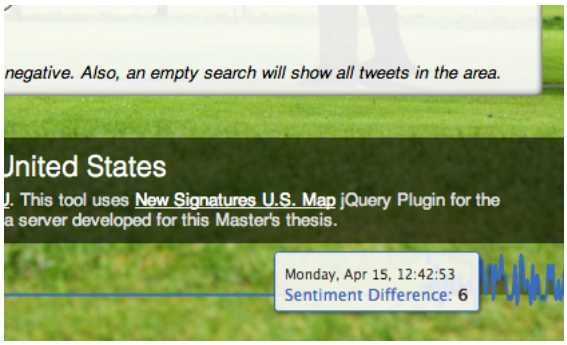


Figure 4.8: Showing the sentiment difference (positive minus negative sentiments) on that exact second from the streamed data.

The SentiMap always starts from scratch on initialize. So refreshing the page or when opening it for the first time, the map has no values and the time line is set to nil. All statistics and colours reflect sentiments from the second a user initializes the application.

As the SentiMap application is designed to be open over a long period of time, the background image of the application changes to a random nature photography every 30 seconds. This to give an extra visual stimuli and to make the application prettier.

4.2.2 SentiGraph: Tweet Sentiment Chart

The Twitter Search API is a popular service that allows for developers to search the Twitter corpus for recent tweets on a given keyword or topic. To show how to use this Search API with our classification server, a demonstration application called SentiGraph was developed. SentiGraph consists of a chart and a Twitter timeline that shows the sentiment for each of the tweets in the chart. The chart is a combination of a pie chart and a bar chart, where both are divided into positive, neutral and negative

tweets. The sentiment of the tweets is indicated with red (negative), blue (neutral) and green (positive) colours.

4.2.2.1 Implementation

SentiGraph is a light-weight web application that runs exclusively in the client's web browser. It consists of three main views: a search field, a combined chart, and a Twitter timeline. The communication between the application and the API Layer is through Asynchronous JavaScript and XML (AJAX) technologies. AJAX is used to request and receive responses from the classification server. By using AJAX for communication, the views can be updated with new data without reloading the entire application.

The tweets shown in the timeline include the tweet text, the date it was published, the author's profile picture and the sentiment classification. All these data are extracted from the tweet JSON object provided by the API layer. The tweets are returned from Twitter API in a chronological order, so when the tweets are rendered they are prepended to the HTML container element to show the most recent tweet first.

A JavaScript library called Highcharts was used to render the combined pie and bar chart. Highcharts offers intuitive and interactive charts written in HTML and JavaScript, and was a good fit for this task.

The application uses CSS for responsiveness, so it fits different screen sizes. On large screens and resolutions, the presentation is divided into two columns, but when the browser window is minimized to below 1080 pixels, the timeline in the right column is moved to the left, so it appears below the chart view.

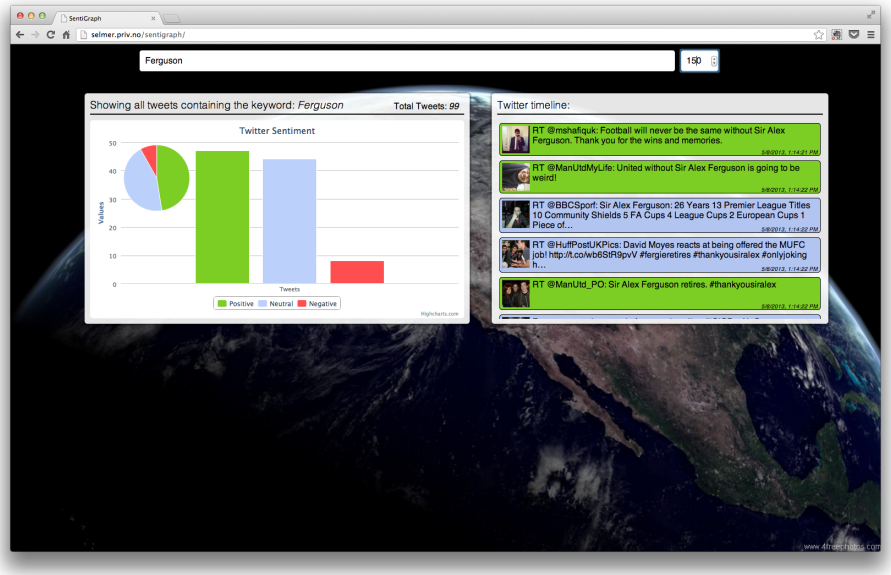


Figure 4.9: Screen shot of the SentiGraph system working, the same day as Manchester United confirmed Sir Alex Fergusons retirement. Showing overwhelming positive tweets.

4.2.2.2 Working Product

SentiGraph allows the user to define a search query and the number of tweets to be returned. The maximum limit supported by the Twitter API is 100 tweets per query. The returned tweets are processed in JavaScript, and for the timeline view, each tweet's HTML container element is given a CSS colour code to indicate its sentiment. This can be seen in figure 4.11. In addition to colouring the container, the script also inserts the author's profile picture, and the date and time the tweet was published. The script waits for all data in each tweet container to be completely downloaded before it renders them in a chronological order. While the application is downloading the data, a loading screen appears in the timeline view. This is mainly to avoid collapsing HTML elements and design flaws because

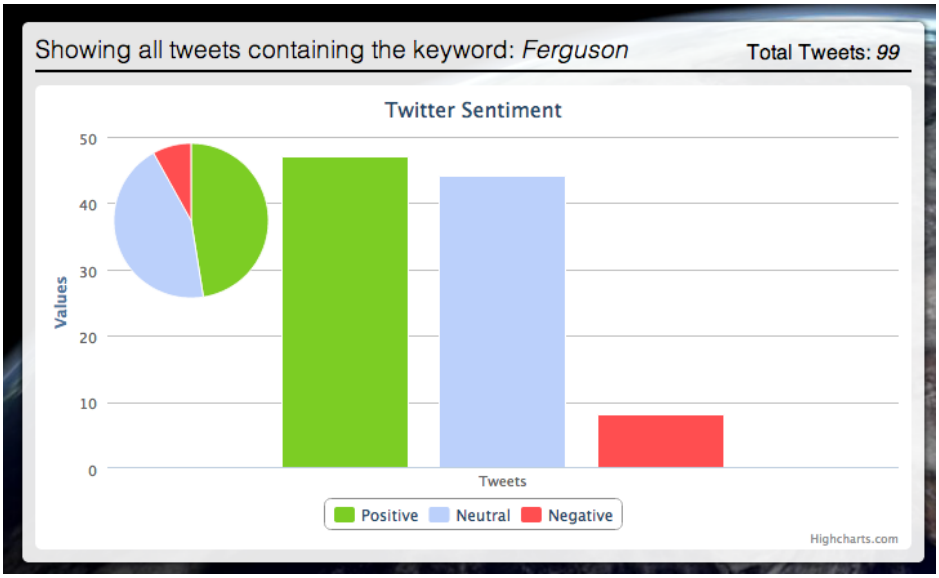


Figure 4.10: Screen shot of the SentiGraph plot close up. The Screen shot is taken the same day as Manchester United confirmed Sir Alex Fergusons retirement. Almost 50% of all the tweets show a positive sentiment, and less than 10% negative.

of missing data, but it is also an indication that the application is busy.

The combined pie and bar chart supports some interaction from the user. When clicking one of the bars in the bar chart, it shows how many tweets that are predicted as the selected class. By clicking one of the sectors in the pie chart it will display the percentage of tweets in that class.

An overview of all the views and the application can be seen in figure 4.9. The top has the query input box, and on the left side the plots are visible (seen in detail in figure 4.10). All the tweets that form the plots can be seen in detail in the timeline displayed in figure 4.11.

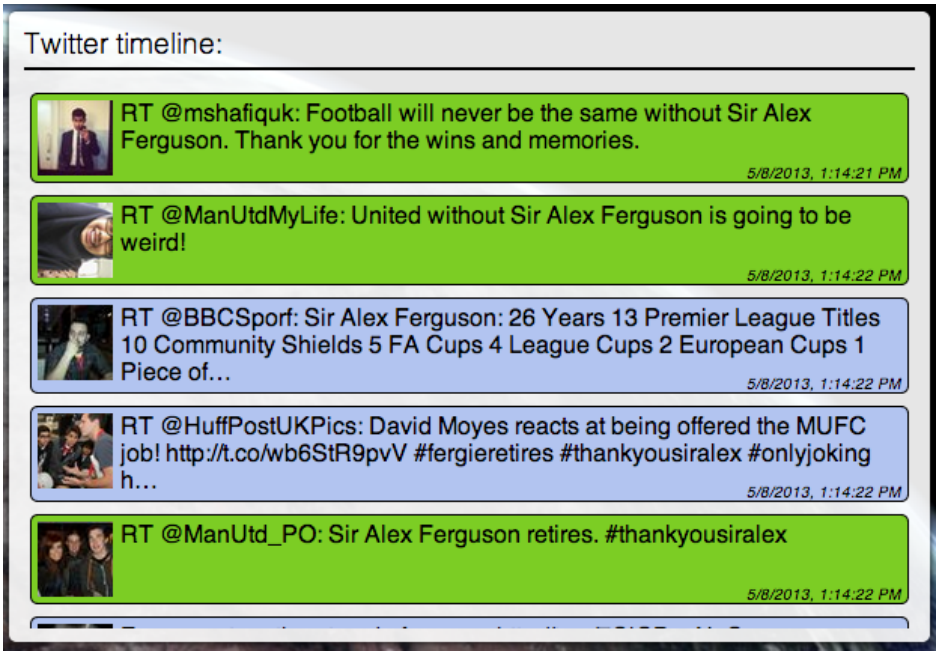


Figure 4.11: Screen shot of the tweet list on the SentiGraph application. Screen shot taken the same day as Manchester United confirmed Sir Alex Fergusons retirement. The tweets shown are an excerpt of the entire list of tweets. The tweets have different background colours according to their sentiment.

4.2.3 SentiStack: Comparing Queries

SentiStack is an application to easily compare sentiments between an arbitrary amount of search queries. SentiStack uses the Twitter Search API to retrieve data, and visualises the data as a bar stack per search query. If there are three queries, three bar stacks will be presented in the graph. A bar stack consists of three values: neutral tweets, positive tweets and negative tweets. See a screen shot of the final product in figure 4.12.

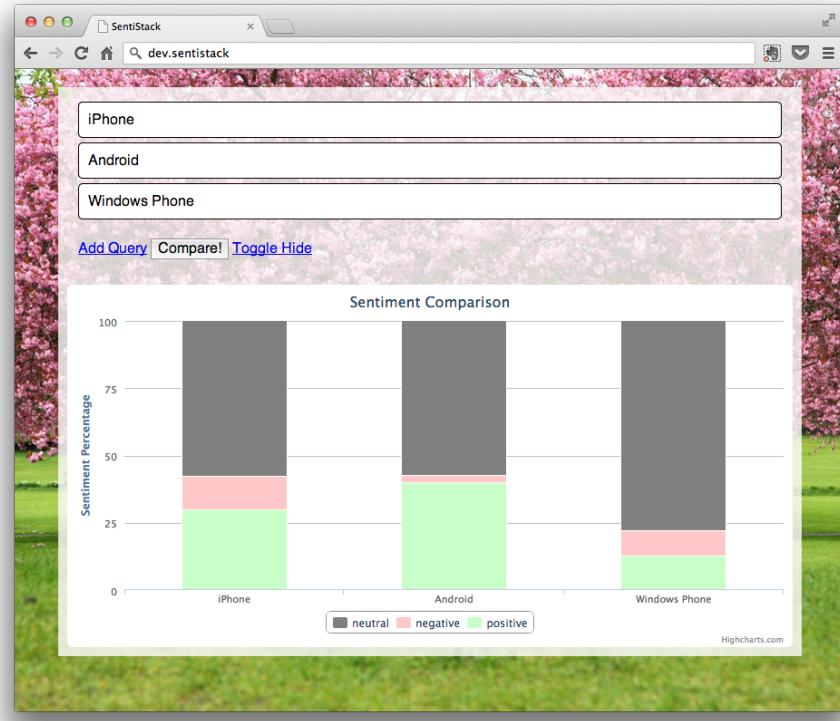


Figure 4.12: Screen shot of the SentiStack system. Showing a comparison of iPhone, Android and Windows Phone. The graph indicates that there are most positive tweets about Android.

4.2.3.1 Implementation

Like SentiGraph, SentiStack is a client side application, and does not require any server side code base. This means that the application can be easily distributed and run locally without any server running to execute the code.

There are two different aspects to the SentiStack code: input queries,

and output graphs. Communication with the input form happens through events. When the SentiStack system is initiated, it listens for three events: a new query is requested, the "toggle input boxes" button is clicked, and the "Compare" button is triggered. If one of these events is triggered, an attached method will respond and execute appropriate code; either append input box, hide input boxes or initiate comparison and present the result as a graph.

When the "Compare" button is pressed, the system extracts each query from the input boxes, and constructs a query list. This query list will be iterated and a total of 200 tweets from each query will be retrieved from the Search API using AJAX. As the API responds in JSON, the data can be easily parsed and manipulated. All results from the queries are accumulated in a hash map using the query as a key and a list of the sentiment distribution for that query (e.g., 70 neutral, 70 positive and 60 negative) as a value. When every API request is done, the presentation method is triggered.

The presentation method uses the accumulated data hash map as a source and generates a graph based on that information. For each key in the hash map, a bar stack is generated with the value as distribution for the parts of the stack. The search queries (hash map keys) are used as labels for the graph.

If the compare event is triggered again, the system will reset (clearing the data hash map and query list) and start the entire process again.

4.2.3.2 Working Product

When opening the SentiStack system, an empty input box is shown, without any graph. The graph will only appear when requested by the "Compare" button. When entering queries in the input boxes and pressing "Compare" the graph will be generated and presented as shown in figure 4.12.

To add a new query to the system, the "Add Query" link is used. When pressing the link, the last input box is cloned and the cloned box is appended to the list without any value. This process is shown in figure 4.13. There are no limitations of the number of queries that can be added, but if there are many of them, the query labels can be hard to read in the graph. If there are many queries, the graph will be pushed down on the page. To avoid this, the "Toggle hide" button can be used. This button toggles the visibility of all except the first input boxes.

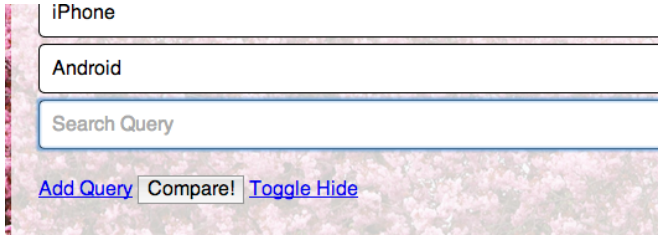


Figure 4.13: Screen shot of the SentiStack system when adding a new query. Figure shows the system after "Add Query" is pressed. The added input box gains focus and is ready to receive query.

Details for the different queries is visible by hovering over a given bar stack in the graph. The details consist of the number of positive, negative and neutral tweets, as well as the percentage for each of these classifications. This detailed view can be seen in figure 4.14.

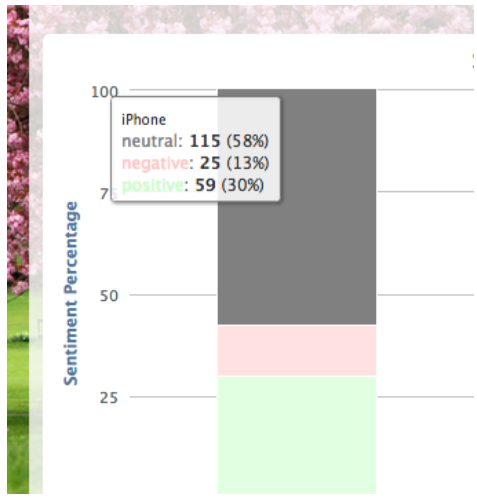


Figure 4.14: Screen shot of the SentiStack system when hovering over a bar stack and showing the details for a given query.

Chapter 5

Experimental Setups

To experiment with different solutions for sentiment analysis, a system testing platform and code base was developed. This testing system generates and trains different models based on input arguments as default values for algorithms and type of algorithm. The architecture and flow of this system is described as a part of section 4.1.2.

The testing system can take in a set of parameters to use for an algorithm, like pre-processor methods, whether or not to use inverse document frequency (IDF) or stop words, and so on, or a grid search flag can be set. If the grid search option is activated, a model is generated with the best possible parameters set for the given algorithm. The grid search is conducted using k-fold cross validation, and the set of parameters to search across. The parameter search space is reflected in table 5.1. The following parameters were included in the search. Three binary (Yes/No) parameters: $UseIDF$, $UseSmoothIDF$, and $UseSublinearIDF$, together with $ngram$ ($unigram$/$bigram$/$trigram$). SVM and MaxEnt models in addition included C and NB models $alpha$ parameters, all with the value ranges [$0.1/0.3/0.5/0.7/0.8/1.0$]. MaxEnt models also had *penalty* ($L1$/$L2$).

	NB	SVM	MaxEnt
penalty	-	-	L1 or L2
alpha/C	<0.1, 0.3, 0.5, 0.7, 0.8, 1.0>		
ngram	Unigram, Bigram or Trigram		
Remove Stop Words	Yes or No		
Use IDF	Yes or No		
Use Smooth IDF	Yes or No		
Use Sublinear IDF	Yes or No		

Table 5.1: Overview of parameter search space for the grid searches conducted in the experiments.

5.1 Pre-processing and Feature Selection

This section describes the different pre-processing implementations used; what placeholders, negation attachment and reducing letter duplications is.

To find the best features to use and what kind of pre-processing was the best, a set of 8 different combinations of pre-processing methods was designed. The different methods include no pre-processing, where all characters are included as features; full remove where all special Twitter features like user names, URLs, hash tags and emoticons are stripped; and one where the Twitter features are replaced with placeholder texts to reduce vocabulary. A full overview of the pre-processing methods is given in table 5.2.

5.1.1 Removing Features

To reduce the vocabulary and to get more precise classification, some features were omitted. User names are most likely to be considered noisy, but in some cases they may be relevant for a text's sentiment. The same goes for URLs, RT-tags and emoticons. Some methods, like P1, P2, P4,

	None	P1	P2	P3	P4	P5	P6	P7
Remove Usernames		x	x		x	x		x
Username placeholder				x				
Remove URLs			x		x	x		x
URL placeholder				x				
Remove Hash-tags					x	x		
Hash-tags as words			x					
Hash-tag placeholder				x				
Remove RT-tag			x		x	x		
Remove emoticons					x	x		
Reduce letter duplicates			x		x		x	x
Attach negation					x		x	x

Table 5.2: Description of the pre-processing methods used for the experiments. Some functions remove entities, other replace them with a placeholder text. The hash-tag as word transforms a hash-tag to a regular word and uses the hash-tag as a feature. "Reduce letter duplicates", reduces redundant letters to a maximum of three.

P5 and P7 consider the user names noisy and remove them from the text. Removing features will remove all notion of the feature, unlike when replacing it with a placeholder.

5.1.2 Replacing with Placeholders

There might be a chance that URLs or user names are relevant for the sentiment. Not necessarily the value of the name or URL itself, but the fact that there are references to URLs and user names. To make these features more informative for the machine learning algorithms, a pre-processing method (P3) was implemented for replacing them with placeholder texts. This means that a user name like $@someuser$ is replaced by the text $||U||$. $||U||$ was chosen as it is very unlikely that it would be a part of the original tweet. Hash-tags ($\#hash$) are replaced by $||H||$ and URLs by $||URL||$.

5.1.3 Reducing Letter Duplication

By reducing and normalizing excessive letter duplications like "I'm soooooo happyyyyyyyy!!!!!!", the vocabulary is reduced and the classification can perform better. However, there might be that the word "sooooo" is more sentimentally charged than the proper form of the word "so". The sentence "I'm soooooooo happyyyyyyyy!!!!!!" is more positive polar than the statement "I'm so happy!".

To reduce the vocabulary but not lose information, all duplicates of more than three consecutive characters are reduced to exactly three duplications. So "I'm soooooooo happyyyyyyyy!!!!!!" would be changed to "I'm sooo happyyy!!!", and the same goes for "I'm sooooooooooooooo happyyyyyyyy!!!!!!". This way the additional sentiment is preserved.

5.1.4 Attaching Negation

The negation word "not" can change the polarity of an entire sentence. The sentence "I am happy" is obviously positive, and "I am not happy" negative. If using unigrams, the features of the second sentence would have been $('I',' am',' not',' happy')$ and for the first one $('I',' am',' happy')$. Both having the word "happy" in them, and if that were to be a significant feature for positive classification, the negative sentence could be classified as positive.

By attaching the negation word to the preceding and the following word, the features will also reflect the change in polarity. So the features for the negative sentence would be $('I',' am - not',' not - happy')$.

5.2 SemEval'13

The system from this Master's Thesis participated in a workshop with focus on semantic analysis systems. SemEval'13[1] had several shared tasks, but we only took part in Task 2B; building a message polarity classification system in Twitter. The task had two parts: unconstrained and constrained systems. The unconstrained system allowed for a training set in addition to the one provided by SemEval'13, while the constrained system could only use the provided set. To test the systems beyond Twitter, SemEval'13 also evaluated the systems using a Short Message (or Messaging) Service (SMS) test set. Thus, there were a total of four different deliveries to SemEval'13: Twitter unconstrained and constrained, and SMS unconstrained and constrained.

5.3 Visualisation Applications

To test the visualisation applications, an experiment to predict the outcome of the Eurovision Song Contest (ESC) was conducted.

Eurovision Song Contest is a popular annual cross-Europe song contest. Each country participates with their selected song, and every country vote for their favourite song to win the contest. A country cannot vote on their own song. In addition to tele-voting in each country, a country also has a panel of judges giving expert opinions.

ESC has historically been criticized for being too political, but research can indicate otherwise. Ginsburgh and Noury [2008] show that the amount of vote trading is rather minimal and that the voting is more cultural and geographical than political. Furthermore, Ginsburgh and Noury [2008] suggest that immigration can have some impact on voting. In any case, it may be plausible that the amount of tele-voters for ESC is equal to the amount of people tweeting about a given country contribution.

[1] `http://www.cs.york.ac.uk/semeval-2013/`

To predict the outfall of ESC, the SentiStack application was used. This allows for several Twitter Search queries to be compared to each other, and thus see what may be considered as favourites. The predictions were carried out before the ESC final, but after the semi-finals.

Chapter 6

Experimental Results

The experimental results is split into four different sections. In the first section, the full grid search results are defined. In the second, a detailed comparison of the best algorithms can be found. Section 6.3 describes the result from SemEval'13, and Section 6.4 covers the ESC experiment.

6.1 Full Grid Search

As described above, an extensive grid search was conducted. This search cycled through different algorithms, parameters and preprocessing techniques. Figure 6.1 displays the precision, recall, F1-score and accuracy for each of the classifiers with dev 1 as evaluation data. We notice that most of the classifiers that involve the NB algorithm have a bad performance, both for accuracy and F1-score. Further, we can see that the MaxEnt classifier has the best accuracy, while SVM has a slightly better F1-score.

Two-step models with SVM-based subjectivity classification exhibit the same basic behaviour. The one-step MaxEnt model classifies more tweets

as neutral than the other classifiers. Using MaxEnt for subjectivity classification and either MaxEnt or SVM for polarity classification performs well, but is too heavy on the positive class. Boosting does not improve and behaves in a fashion similar to two-step MaxEnt models. All combinations involving NB tend to heavily favour positive predictions; only the two-step models involving another algorithm for polarity classification gave some improvement for negative tweets.

We can also see that all the classifiers with SVM tend to give a better confusion matrix than the others. This is shown in Figures 6.2 - 6.14. The figures show what classifications that were correct. The columns are [$negative$, $neutral$, $positive$]. This means that, in the figures, the top left box is how many tweets that were classified as negative, and actually were negative, the box to the right is how many tweets were classified as neutral, but was negative, etc. The blue colour indicate low number of tweets, and the red colour indicate a high number. What we would like to see is a red colour on the diagonal. The confusion matrices that include SVM indicate that it has a more precise classification of negative instances than the others. In tests where NB is used, the system leans much more to the positive class than the other models.

As a part of the grid search, the system applied all the different preprocessing methods for each classifier. Figure 6.15 shows that $P2$ (removing user names, URLs, hash- tags prefixes, retweet tokens, and redundant letters) is the preprocessing method which performs best (gives the best accuracy) and thus used most often (10 times). Figure 6.15 also indicates that URLs are noisy and do not contain much sentiment, while hashtags and emoticons tend to be more valuable features ($P2$ and $P7$ — removing URLs — perform best, while $P4$ and $P5$ — removing hashtags and emoticons in addition to URLs — perform badly).

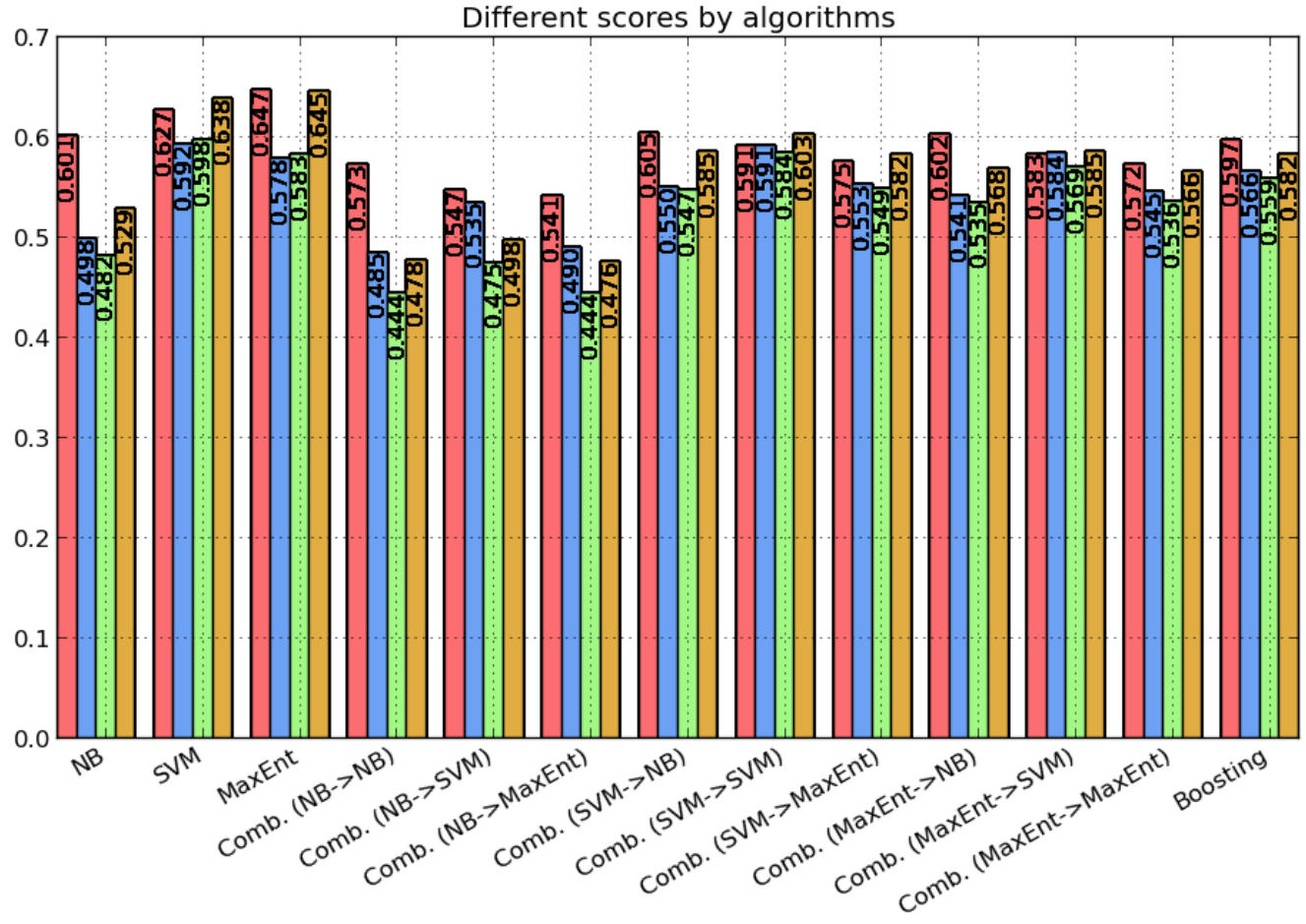


Figure 6.1: A full overview of the performance of the different models after a full grid search. Two models stand out as having highest accuracies: SVM and MaxEnt. One can also see that models using Naïve Bayes, especially on its own or to classify subjectivity, perform poorly.

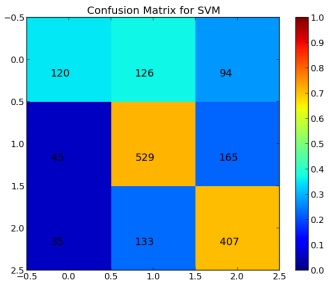


Figure 6.2: Confusion matrix for the model using SVM. Performs well on both neutral and positive predictions, but somewhat poorly on negative.

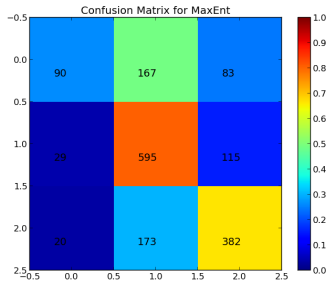


Figure 6.3: Confusion matrix for the model using MaxEnt. Performs especially good on neutral tweets, but seems to be classifying neutral too much.

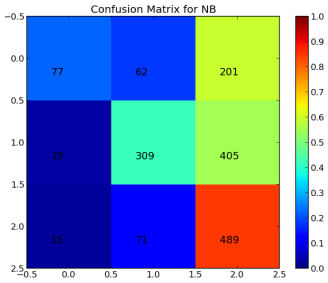


Figure 6.4: Confusion matrix for the model using NB. Performs very well for positive tweets, but seems to favour them too much.

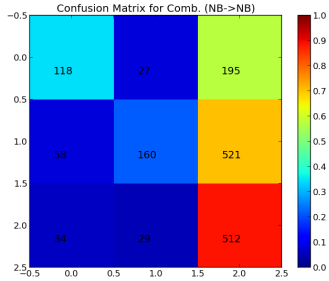


Figure 6.5: Confusion matrix for the two-step model using NB for both subjectivity/objectivity and polarity classification. Shows the same trend as when using NB in a one-step model: the model favours positive predictions.

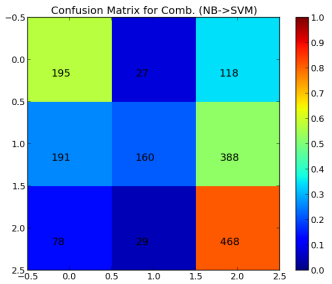


Figure 6.6: Confusion matrix for the two-step model using NB for subjectivity/objectivity classification and SVM for polarity. Shows the same trend as when using NB in a one-step model: the model favours positive predictions but is performing better for negative tweets.

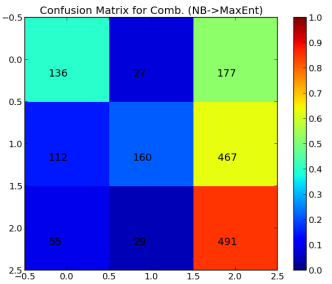


Figure 6.7: Confusion matrix for the two-step model using NB for subjectivity/objectivity classification and SVM for polarity. Shows the same trend as when using the NB -> SVM model: the model favours positive predictions but is performing better for negative tweets.

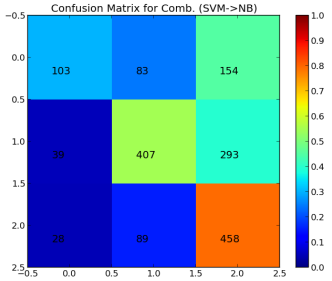


Figure 6.8: Confusion matrix for the model using SVM and NB. Performs well for positive tweets, but not as well for neutral and negative.

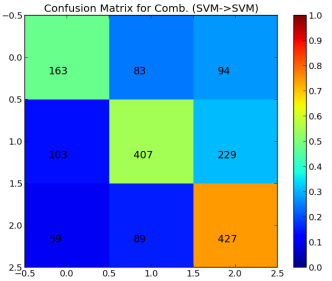


Figure 6.9: Confusion matrix for the model using SVM and SVM. Performs well across the board, and shows a good diagonal colour profile in the plot.

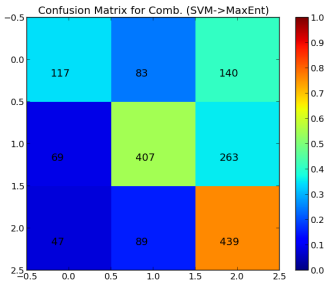


Figure 6.10: Confusion matrix for the model using SVM and MaxEnt. Performs well for neutral and positive tweets, but not too well on negative tweets.

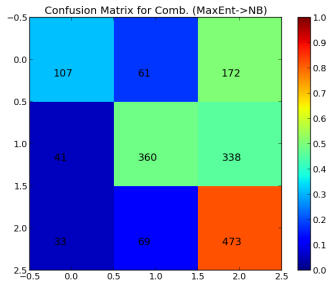


Figure 6.11: Confusion matrix for the model using MaxEnt and NB. Performs well for positive tweets, but not too well on negative and neutral tweets.

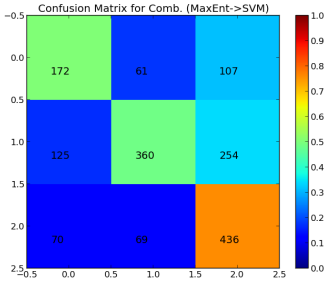


Figure 6.12: Confusion matrix for the model using MaxEnt and SVM. Performs well but is too heavy on the positive classification.

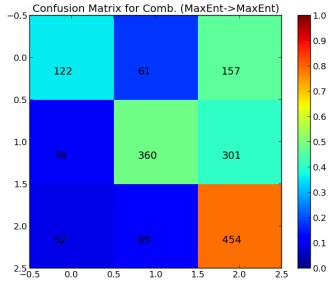


Figure 6.13: Confusion matrix for the model using MaxEnt and MaxEnt. As with MaxEnt -> SVM, this model performs well but is too heavy on the positive classification.

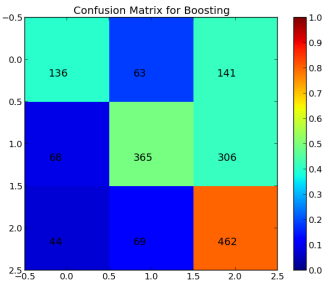


Figure 6.14: Confusion matrix for the model using Boosting. Showing the same trend as the MaxEnt -> SVM and MaxEnt -> MaxEnt models: it tends to classify too many tweets as positive.

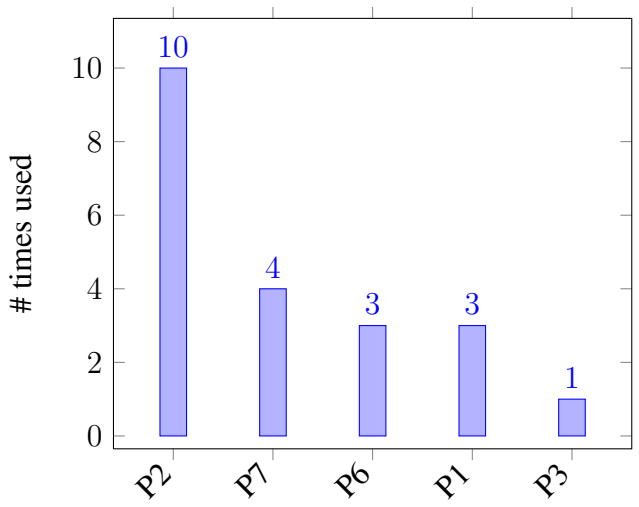


Figure 6.15: Statistics of pre-processing usage. Removing all usernames, URLs, hash-tag characters, RT-tags and excessive letters seem to give best performance.

	SVM	MaxEnt
ngram_range	1,1	1,1
sunlinear_tf	True	True
preprocessor	P2	P3
use_idf	True	True
smooth_idf	True	True
max_df	0.5	0.5
stop_words	None	None
C	1.0	0.3
penalty		L1

Table 6.1: The parameters selected for SVM and MaxEnt models through extensive grid search.

6.2 SVM vs MaxEnt

From the full grid search, the best parameters on SVM and MaxEnt were extracted and more detailed tests were carried out only using SVM and MaxEnt, without grid searching. The best parameter setting for the SVM model was using unigram, the $P2$ preprocessing method, not removing stop words, and a C value of 1.0. For the MaxEnt model, the $P3$ preprocessing method was used, along with unigrams, IDF, not removing stop words, and 0.3 as C value. The penalty for MaxEnt was selected to be L1. Full parameter selection for SVM and MaxEnt is visible in table 6.1. The SVM and MaxEnt models, with these parameters, were tested with an additional development set (the data set Dev 2; see table 3.1).

The results for both the SVM and MaxEnt classifiers with Dev 1 are shown in figure 6.16. With Dev 2, SVM's performance is much better than MaxEnt's, as seen in table 6.2. Dev 1 contains more neutral tweets than Dev 2, which gives us a reason to believe that it favours the MaxEnt classifier. In MaxEnt's confusion matrix from the first search, shown in Figure 6.3, we can see that it classifies more neutral tweets than the other classifiers.

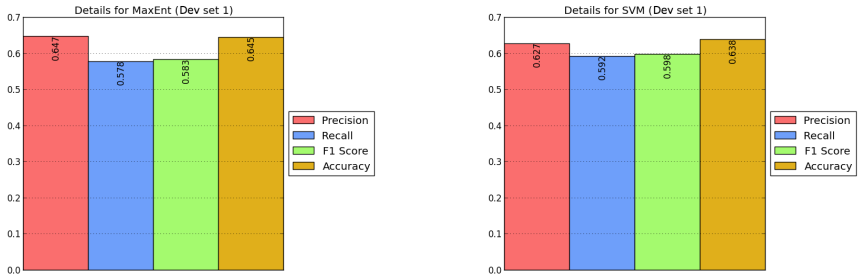


Figure 6.16: Best performance plots for SVM and MaxEnt. MaxEnt has better Precision and Accuracy, but SVM scores higher on F1-score and Recall.

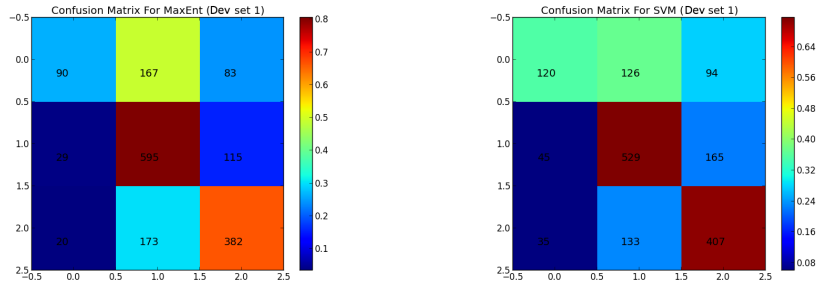


Figure 6.17: Confusion Matrix for SVM and MaxEnt. MaxEnt favours neutral classification, and performs worse than SVM on both positive and negative classifications.

To even out the distribution of the different classes, we did a grid search with a reduced training set. Figure 6.21 and 6.20 show SVM's results when reducing the training set to a maximum of 1000 and 2000 tweets per class. The accuracy of SVM is highest on the original training set, but the F1-score and Recall improves when reducing the training set to maximum 2000 per class. See complete effects of reducing training set in table 6.3. The negative classification improves when reducing the training set to maximum 2000 tweets per class, but the accuracy for neutral and

Data set	Dev 1		Dev 2	
Learner	SVM	MaxEnt	SVM	MaxEnt
Precision	0.627	**0.647**	**0.700**	0.561
Recall	**0.592**	0.578	**0.726**	0.589
F1-score	**0.598**	0.583	**0.707**	0.556
Accuracy	0.638	**0.645**	**0.728**	0.581

Table 6.2: Best classifier performance (bold=best score). Shows that using Dev 1, MaxEnt has better accuracy and precision, but SVM performs better in regards to F1-score and recall. For Dev 2, SVM has higher measures across the board.

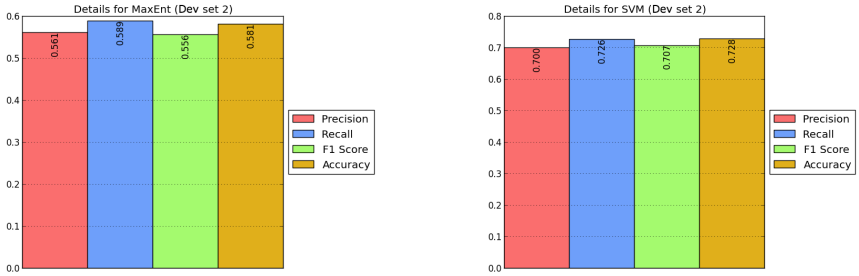


Figure 6.18: Best performance plots for SVM and MaxEnt using Dev 2. SVM outperforms MaxEnt with **0.728** in accuracy versus **0.581**. SVM performs better according to all measures.

positive tweets decrease. Similar observation can be made when reducing the data set to 1000. Negative tweets perform better, but positive and neutral performance are worse. This can be deduced from figures 6.17, 6.20 and 6.21.

The features in table 6.4 are the most informative features from each of the classifiers in the second grid search. Some features are represented among the top 15 for both SVM and MaxEnt, and most of these features make

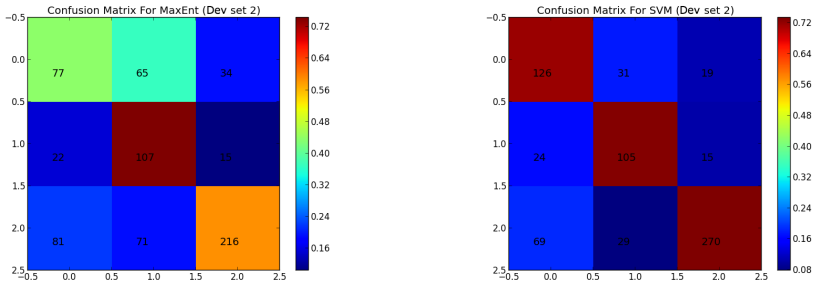


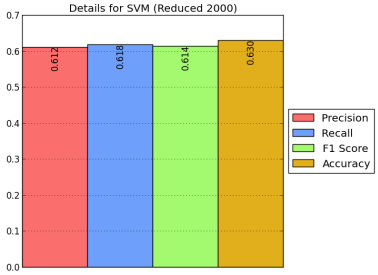
Figure 6.19: Confusion Matrix for MaxEnt and using dev set 2. The performance difference between SVM and MaxEnt is visible in the confusion matrices as well. SVM shows a solid diagonal colour.

Data size	Original	2000	1000
Precision	**0.627**	0.612	0.570
Recall	0.592	**0.618**	0.586
F1-score	0.598	**0.614**	0.563
Accuracy	**0.639**	0.630	0.569

Table 6.3: Effects on SVM trained with reduced training sets

sense. As we did not normalize the features, we can see that some words appear in different forms and degrees (e.g., "worse", "worst" and "don't", "didn't"). For both MaxEnt and SVM, emoticons are listed among the most informative features. In the case of SVM, the exclamation mark is an informative feature for positive classification.

We see that for the most part the features seem to make sense, but we see some anomalies. Both SVM and MaxEnt show 'why' as a negative feature, which is normally incorrect. In addition, the most informative positive feature for SVM is 'wait', which is not a particularly positively charged word.



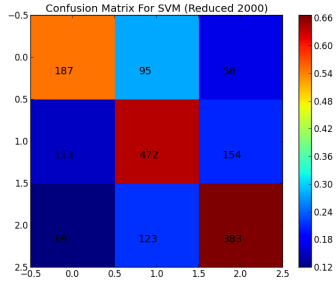
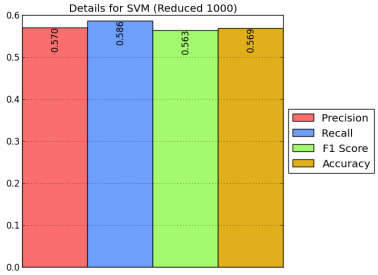


Figure 6.20: Performance of SVM when reducing the dataset to maximum 2000 per class. Showing better recall and F1-Score than without reduction, but poorer accuracy and precision. Somewhat better performance on negative tweets, but poorer on neutral and positive.



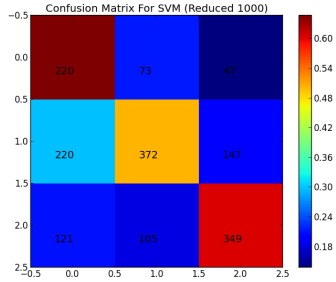


Figure 6.21: Performance of SVM when reducing the dataset to maximum 1000 per class. Showing better performance on negative tweets, but worse on neutral and positive. Performance is generally poorer than without reduction.

SVM			MaxEnt		
Negative	Neutral	Positive	Negative	Neutral	Positive
no	wear	wait	no	center	glad
didn't	tallahassee	nice	don't	royal	nice
cancelled	15th	interesting	why	plan	thanks
don't	26	awesome	bad	joe	cool
worse	at	cool	shit	nov	awesome
why	trip	amazing	injury	george	interesting
bad	joe	fun	cancelled	theres	amazing
worst	theres	!	hate	arrows	:)
hate	arrows	:)	worst	trip	fun
shit	murphy,	excited	worse	set	best
sad	question	happy	fuck	question	excited
sorry	plan	love	sorry	at	love
not	set	best	not	url	happy
fuck	8th	great	sad	8th	good
:(	paterno	good	:(	paterno	great

Table 6.4: Top 15 of the most informative features for SVM and MaxEnt

	Twitter		SMS	
NTNU-	NTNUC	NTNUU	NTNUC	NTNUU
Precision	**0.652**	0.633	**0.659**	0.623
Recall	**0.579**	0.564	**0.646**	0.623
F1-score	**0.590**	0.572	**0.652**	0.623
F1 pos/neg	**0.532**	0.507	**0.580**	0.546

Table 6.5: **NTNUC** and **NTNUU** in SemEval'13

6.3 SemEval'13 Results

Based on the information from the grid search, two systems were built for SemEval'13. Since one-step SVM-based classification showed the best performance on the training data, it was chosen for the system participating in the constrained subtask, NTNUC. The pre-processing was also the one with the best performance on the provided data, P2 which involves lower-casing all letters; reducing letter duplicates; using hash-tags as words (removing #); and removing all URLs, user names and RT -tags.

Given the small size of the in-house ('NTNU') data set, no major improvement was expected from adding it in the unconstrained task. Instead, a radically different set-up was chosen to create a new system, and train it on both the in-house and provided data. NTNUU utilizes a two-step approach, with SVM for subjectivity and MaxEnt for polarity classification, a combination intended to capture the strengths of both algorithms. No preprocessing was used for the subjectivity step, but user names were removed before attempting polarity classification.

As further described by Nakov et al. [2013], the SemEval'13 shared task involved testing on a set of 3813 tweets (1572 positive, 601 negative, and 1640 neutral). In order to evaluate classification performance on data of roughly the same length and type, but from a different domain, the

evaluation data also included 2094 Short Message Service texts (SMS; 492 positive, 394 negative, and 1208 neutral).

Table 6.5 shows the results obtained by the NTNU systems on the SemEval'13 evaluation data, in terms of average precision, recall and F-score for all three classes, as well as average F-score for positive and negative tweets only (F1 + / –; i.e., the measure used to rank the systems participating in the shared task).

The NTNU system ranked at 24th of 36 on constrained and 10th of 15 on unconstrained. For the SMS task, the NTNU system ranked 5th of 28 constrained systems and 6th of 15 unconstrained.

6.4 Visualisation Results

During the experiment a screen shot of the SentiStack system was taken to document the results (see figure 6.22).

The application was run before the final, but after the semi-finals. The goal was to predict as much of the top-10 list as possible. By sheer guesswork, one could expect to get $\frac{10}{26} = 0.3846 \approx 38.5\%$[1] of the top-10 finalists.

In figure 6.22, the prediction results is shown. By the graph, it is clearly visible that Ireland was predicted to be the winner, followed by Germany, Malta, Finland and Denmark.

Table 6.6 shows an overview of the predictions and the actual results of the ESC 2013. Only 5 of the predicted top-10 actually resulted amongst the top-10; thus the system had an accuracy of 50%.

[1] 10 out of 26 participants.

Place	Prediction	Result
1.	Ireland	**Denmark**
2.	Germany	**Azerbaijan**
3.	**Malta**	**Ukraine**
4.	Finland	Norway
5.	**Denmark**	Russia
6.	**Ukraine**	Greece
7.	Iceland	Italy
8.	Sweden	**Malta**
9.	**Hungary**	Netherlands
10.	**Azerbaijan**	**Hungary**

Table 6.6: The results of the Eurovision Song Contest versus the predictions from using the SentiStack visualisation application. From the table, it is visible that 50% of the countries predicted in the top-10, resulted in a position among the top-10. The countries shared between the lists are marked in bold.

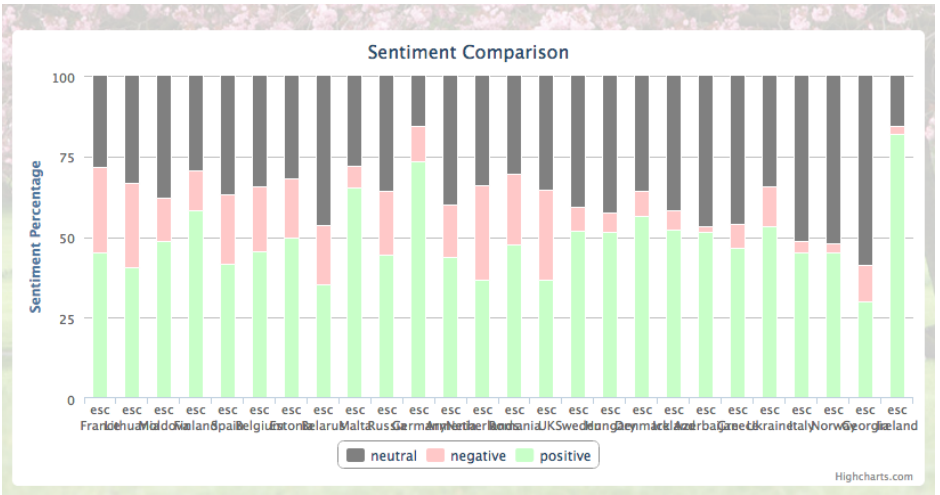


Figure 6.22: Screen shot of the SentiStack application running before the Eurovision Song Contest Final. The order of the bar stacks is equal to the running order: France, Lithuania, Moldova, Finland, Spain, Belgium, Estonia, Belarus, Malta, Russia, Germany, Armenia, The Netherlands, Romania, United Kingdom, Sweden, Hungary, Denmark, Iceland, Azerbaijan, Greece, Ukraine, Italy, Norway, Georgia and Ireland.

Chapter 7

Discussion

In the introduction two main goals were introduced for this Master's Thesis:

G1 Experiment with different models for doing sentiment analysis

G2 Develop tools for visualising sentiment classified tweets

This chapter will discuss whether or not we succeeded in reaching these goals, and the solutions in general. The first section handles G1 and the second section discusses G2. The generic system architecture is an important part of developing the visualisation applications, and thus it is included in the latter section.

7.1 G1: Experiment with different models for doing sentiment analysis

In this section we will discuss whether we succeeded with our first goal or not, and how it was conducted. The first goal stated that we should

try different models for classifying sentiment on short messages, such as tweets.

The experiment description from section 5 shows that we used grid searching on different machine learning algorithms and combinations of algorithms to classify sentiment. A total set of 13 different models were thoroughly tested with a large training set. Our system generated graphs and plots comparing the different models, both on accuracy, F1-score, recall, precision and their confusion matrices. This comparative view allowed us to find out which model performed best.

In our experiments we found that MaxEnt and SVM perform best in terms of accuracy. By conducting the experiments as stated, and finding the models with the highest accuracy, we succeed with the first goal.

The data that was used to train and evaluate the different models was given by SemSeval'13, that also provided a test set with SMS messages for the second part of the task. We considered this part of the task an opportunity to test if the system really was domain semi-independent. The results from the task indicate that the system performed well both for the constrained and the unconstrained versions. The system ranked 5th of 28 constrained systems and 6th of 15 unconstrained systems on SMS classification, which is an encouraging result in terms of domain semi-independence. A part of the strategy to obtain good classifications on several domains was to use general feature selection methods. When looking at our feature selection for the SMS constrained, we see that no Twitter specific features are used. User names, hash-tags as words, URLs, RT-tags, etc, are removed. What we have left is essentially the same content as an SMS would have; text with emoticons. For the unconstrained system, some Twitter specific features are included, such as hash-tags, RT-tags and URLs, but even with these features a tweet is similar to an SMS message. Both tweets and SMS messages are short (respectively 140 and 160 characters), and contain similar language.

Some of the preprocessing methods are to be considered naïve, such as the negation handling. In these experiments we only utilized unigram

feature selection with a simple negation support. This simple method binds all appearances of the word 'not' to the next and previous word in the sentence, e.g., 'is not bad' would be merged into the unigram ['is-not', 'not-bad']. This is a very naïve approach which in some cases may extend the vocabulary with little informative features.

From the listed features, emoticons are among the most informative, both for SVM and MaxEnt. However, we did not use any placeholders to normalize these emoticons. By using placeholders, the emoticons that are not frequently used would gain information. Another potential improvement is support for Emoji[1], which have very similar semantics as emoticons, but are implemented in Apple products as an own character set (rather than constructed from ASCII characters such as ':)' and ':D', etc). There are reasons to believe that these Emojis would provide informative features, and could either be converted to ASCII characters or placeholders to merge them with the emoticons. When not supported, the Emojis would show up as ASCII squares: '□'

While the focus for this goal was to experiment with different models for SA, there are still some interesting approaches that remain unexplored.

The data sets used to train our models were not evenly distributed among the different target classes. This may have affected the results for some of the algorithms that were used. The data has a large amount of neutral tweets, which seemed to favour especially MaxEnt when classifying neutral tweets. To even out the distribution in the data set, we tried to limit the number of tweets per class. The results when limiting to 2000 tweets per class gave better recall and F1-measure, but a small decrease in both accuracy and precision. The confusion matrix also indicates better ability to successfully classify negative instances. This was an expected result since the classifier was trained on a more balanced data set. But the data set was still missing some (about 800) negative training instances to be perfectly balanced. The decreased accuracy and precision may be a consequence of lacking training data across all classes. This theory is backed

[1] `http://en.wikipedia.org/wiki/Emoji`

up by the decreasing performance when limiting to 1000 tweets per class, and the increasing accuracy when classifying negative tweets.

7.2 G2: Develop tools for visualising sentiment classified tweets

We achieved goal 2 by implementing three different applications for visualising sentiment analysis data, and by designing and using a generic system for classifying tweets.

We are satisfied with the way the generic system architecture is built. By just extending the Twitter API, no API documentation is needed, and if you are familiar with the Twitter API, you do not have to re-learn anything. In addition, if there is a system allready integrated with Twitter data, the migration to our system is simple: just swap the entry URL point from the Twitter API base URL to our system's base URL.

The current implementation of the API Layer is simplified with regards to authentication. The API Layer has no OAuth server, but rather uses its own credentials to connect to Twitter. This means, if 10 different users do 30 requests each, Twitter's request limit will engage and our API Layer will be put on hold until next window of requests. A better solution would have been to implement a mirror of Twitter's OAuth server and pass on the received credentials to Twitter. This way each end-user or application client would have their own pool of requests.

One important point when designing the generic system was to make it as fast as possible. This to be able to handle large amounts of tweets when streaming or simply searching with a high count limit. In some cases a client can request 1000 tweets at once. This required a system that can operate in parallel and handle asynchronous connections. The way our system was built, the API layer works independently of the classification server and each request is parallel and asynchronous. This maximizes the

number of tweets we can handle and reduces the collected wait time for the client. This solution works great for the applications we have built and for up to 5 clients running simultaneously, but the system has not been tested with any more clients or stress-tested in any way. A stress-test might show a bottleneck or a weakness in the system that is not apparent at this point.

7.2.1 SentiMap

The SentiMap application shows interesting information and distribution of tweets across the USA. It is capable of handling at least 50 tweets per second, and updates the map's colour scheme for each tweet. Every tweet is grouped by state, so the system looses some information with regards to locality. It could have been interesting to add more details to the map, showing the exact origin of a tweet in addition to it changing the state sentiment indication colour.

When opening SentiMap now, you start out from scratch. There is no history or storage for the tweets. If you were to open SentiMap and have it classify 1000 tweets, refreshing the application will remove these 1000 tweets. Adding a local storage for these tweets, could provide some usefulness. Also concatenating search data with stream data could be useful, starting the application with some initial data. If some big event happens now, a user has to be quick to open SentiMap to see the sentimental development.

There is no automatic way of plugging in a different country to the SentiMap application, but the architecture allows for easy system extension. By having defined a clear MV* architecture, each module is fairly independent of each other, and can thus be replaced with different modules. To make this even easier, a plug-in system could have been designed and documented.

By using WebSockets to provide data from the API Layer, a continuous connection is established. This means that if the API Server goes down,

or the visualisation application back-end server restarts, the application will automatically reconnect without any user action. E.g., if you open SentiMap on a laptop, closes this laptop, and then re-open it, SentiMap will reconnect and start showing data again. This means that you can have the client open over a long, long time, if necessary.

7.2.2 SentiGraph

SentiGraph was developed as a light-weight JavaScript application that runs entirely in the client's web browser. This makes the application both fast and compatible with most platforms. It demonstrates how the API Layer (and the Twitter API) can be used to search for different topics or keywords. It would, however, be possible to combine this with a streaming service to append incoming tweets to the search result. But that would require a small server side application to push the streamed tweets to the client, and thus break the concept of having the entire application running in the browser.

SentiGraph is a good application for visualising the sentiment data that are generated by the sentiment classifier. However, there are definitely room for more features, such as comparison of keywords and, of course, several different graphs and charts. A graph that showed changes in sentiment over time could be a useful tool to visualise trends, and possibly changes caused by certain events or happenings.

7.2.3 SentiStack

SentiStack gives a good comparative view between several different Twitter search queries. Without defining any domain for the queries, the application can be used to compare anything, from products to ESC participation songs. This can also be its downfall, as it might get too generic and not perform as well as an application specifically designed to do product comparisons.

A big limitation of the SentiStack system is the number of requests to the Twitter API. The Twitter Search API has a limitation of the number of requests that is allowed each 15 minutes. SentiStack does two requests per query (to retrieve the total of 200 tweets). This means that if there are a total of 25 queries, 50 requests will be made each time "Compare" is pressed. Only 180 requests can be made every 15 minutes. So when having 25 queries, one can only update the comparative view 3 times every 15 minutes. To solve this problem, a more sophisticated OAuth solution is required on the API Layer. The API Layer needs to receive and pass along OAuth credentials to the Twitter API. This way the request limitations will be per end-user and not global amongst the users of the API Layer.

7.2.3.1 Predicting ESC

By looking at the results from predicting the winners of ESC, it is clear that SentiStack did not perform too well. Only 5 of the finalists predicted to be among the top 10 were correct. This is marginally better than the baseline of 38.5%, but not enough to draw any conclusions.

Ireland was picked as a clear winner by SentiStack, but in fact ended up in decidedly last place with only 5 points.[2] It is difficult to say why Ireland did so poorly when it shows a large amount of positive tweets. It could seem like the assumption of tweets being representative for the tele-voting on ESC is wrong. One important aspect is that since Ireland is an English speaking country, their tweets are visible to the SenitStack system, unlike most of the other countries. This can lead to many biased tweets being registered from Ireland and thus skewing the results. Since people in one specific country are not allowed to vote for that country's song, it would make sense to geographically filter the tweets per country.

SentiStack only fetches 200 tweets per query, this is because of the limi-

[2] `http://en.wikipedia.org/wiki/Eurovision_Song_Contest_2013`

tation of the number of requests from the Twitter API. This might be too few tweets to get a representative picture to predict or analyse statistical outcomes. A better solution might be to mine for data over a long period of time using the Stream API.

It may seem as SentiStack is too generic to predict the outcome of a contest as complex as ESC. A more sophisticated application specifically designed to predict the outcome of ESC could have used prior knowledge and expert opinions in addition to sentiment analysis, and accumulate opinion data over a larger period of time. ESC is not, as mentioned, entirely up to tele-voters, and relies on a panel of judges for 50% of the votes[3]; this has a large impact on the results.

[3] `http://en.wikipedia.org/wiki/Voting_at_the_Eurovision_Song_Contest`

Chapter 8

Conclusion

We tested three different base machine learning algorithms: Naïve Bayes, SVM and MaxEnt, in addition to combining them for doing two-step classification and boosting with voting for sentiment. From experimenting with these setups, we found that machine learning algorithms perform well for domain semi-independent classification of sentiment.

Our experiments show that SVM and MaxEnt in a single step classification performs best. Given the confusion matrices and results on different data sets, it seems that SVM performs better than MaxEnt. But the neutral-heavy development set caused MaxEnt to perform better than SVM while doing a complete full grid search across multiple values.

Notably, both systems perform well on the out-of-domain data represented by the SMS messages, which is encouraging and indicates that the approach taken really is domain semi-independent. This was also reflected in the rankings of the two systems in the shared task of SemEval'13: both were on the lower half among the participating systems on Twitter data (24th/36 resp. 10th/15), but near the top on SMS data, with the constrained system being ranked 5th of 28 and the unconstrained 6th of 15.

As emoticons were proven informative features, support for Emojis as feature would be an improvement. This would give better performance to applications like SentiMap, that processes only tweets with a geolocation attached to it, i.e., it supports only tweets sent from a mobile device, and possibly an Apple product that uses these Emojis.

By reducing the training set, we obtained better classification for negative and positive tweets separately, but when the number of instances was reduced to a perfect balance between positive, neutral and negative classes, the total number of tweets was too low to train a well-performing classifier. A larger dataset would reveal if training with more balanced data would give a better performing classifier.

The architecture proposed for this system performs well on regular REST API requests, but is also capable of streaming and classifying several tweets per second in real-time. By extending the Twitter API and having the same end point interface, no additional documentation is needed for the API. Existing libraries and programming language specific API wrappers can be used to retrieve information from our API Layer by only changing their URL connection point.

We found that the sentiment data can be used for visualisation in different practical views. Since Twitter has a lot of meta data attached to each tweet from the REST API, the possibilities for making visualisation applications are huge. With relatively simple steps, one can make applications for showing accumulated sentiments based on a search query, or show changes in sentiment across a country.

The SentiStack visualisation application can be used to compare sentiments on different queries, but seems too generic to handle predictions of such a complex contest as Eurovision Song Contest. ESC has many different aspects as some voting trading and panels of judges that make it difficult to predict the outcome by only using Twitter data. The SentiStack system also just uses 200 tweets per query, to accommodate for the Twitter API request limit, which seems too little to be representative.

8.1 Contributions

In this project we have defined the state-of-the-art for Twitter Sentiment Analysis systems, and implemented a generic system with an architecture capable of classifying many tweets per second in a live stream. We found that, by extending the Twitter API and just attaching a value for sentiments, the data can easily be used to implement visualisation applications, or extending existing Twitter-based systems.

We have tested different machine learning algorithms, reported to work well with Twitter Sentiment Analysis, and found that SVM and MaxEnt performs well on domain semi-independent short messages.

The development of the visualisation applications shows how Twitter sentiment can be used to get opinions on different topics, keywords or from certain locations. It also shows how the combination of the Twitter API and a generic architecture can be used to easily develop different applications that utilize the data produced by the classification system.

8.2 Future Work

An obvious way to extend this work would be to add other classification algorithms to the grid search, e.g., Conditional Random Fields or more elaborate ensembles. There are also several features and feature selection methods that could be investigated, such as POS-tagging, like Pak and Paroubek [2010], and a less naïve way of handling negation. Rather than the simple treatment of negation used here, an approach to automatic induction of scope through a negation detector [Councill et al., 2010] could be used. Relational features could also be added, as shown by Karlgren et al. [2010] and Johansson and Moschitti [2012].

To improve the classification on tweets, one can make the system less domain independent by adding more Twitter-specific features, e.g., by utilizing automatic phrase-polarity lexicon extraction [Velikovich et al.,

2010], or by adding lexica like the AFINN word list by Nielsen [2011] that is developed specifically for social media or the Twitter-oriented word lists created by Mohammad et al. [2013].

While developing the visualisation application SentiMap, a wide use of Emojis was shown in the tweet stream. As the tweets were restricted by geographical location, most of them originated from hand held devices, such as Apple's iPhone. By translating these Emojis to common placeholders like $||Happy||$ or $||Sad||$, they could be used as features.

Bibliography

Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38. Association for Computational Linguistics, 2011.

All Twitter. Twitter Now Seeing 400 Million Tweets Per Day, Increased Mobile Ad Revenue, Says CEO, 2012. URL `http://www.mediabistro.com/alltwitter/twitter-400-million-tweets_b23744`.

Amir Asiaee T, Mariano Tepper, Arindam Banerjee, and Guillermo Sapiro. If you are happy and you know it... tweet. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1602–1606. ACM, 2012.

Younggue Bae and Hongchul Lee. A sentiment analysis of audiences on twitter: who is the positive or negative audience of popular twitterers? *Convergence and Hybrid Information Technology*, pages 732–739, 2011.

Luciano Barbosa and Junlan Feng. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 36–44. Association for Computational Linguistics, 2010.

Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999. ISSN 0885-6125. doi: 10.1023/A:1007515423169. URL http://dx.doi.org/10.1023/A%3A1007515423169.

Adam Bermingham and Alan F Smeaton. Classifying sentiment in microblogs: is brevity an advantage? In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1833–1836. ACM, 2010.

Asli Celikyilmaz, Dilek Hakkani-Tur, and Junlan Feng. Probabilistic model-based sentiment analysis of Twitter messages. In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 79–84. IEEE, 2010.

Wilas Chamlertwat, Pattarasinee Bhattarakosol, Tippakorn Rungkasiri, and Choochart Haruechaiyasak. Discovering Consumer Insight from Twitter via Sentiment Analysis. *Journal of Universal Computer Science*, 18(8):973–992, 2012.

Murphy Choy, Michelle LF Cheong, Ma Nang Laik, and Koo Ping Shung. A sentiment analysis of Singapore Presidential Election 2011 using Twitter data with census correction. *Social Science Computer Review*, 2011.

Mauro Cohen, Pablo Damiani, Sebastian Durandeu, Renzo Navas, Hernán Merlino, and Enrique Fernández. Sentiment analysis in microblogging: a practical implementation. In *XVII Congreso Argentino de Ciencias de la Computación*, 2011.

Isaac G. Councill, Ryan McDonald, and Leonid Velikovich. What's great and what's not: learning to classify the scope of negation for improved sentiment analysis. In *Proceedings of the workshop on negation and speculation in natural language processing*, pages 51–59, 2010.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 241–249. Association for Computational Linguistics, 2010.

Tristan Fletcher. Support Vector Machines Explained. *Tutorial paper., Mar*, 2009.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. Technical report, DTIC Document, 2010.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for Twitter: annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

Victor Ginsburgh and Abdul G. Noury. The eurovision song contest. is voting political or cultural? *European Journal of Political Economy*, 24(1):41–52, 2008.

Alec Go, Lei Huang, and Richa Bhayani. Twitter sentiment analysis. volume 17, 2009.

Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in Twitter: a closer look. In *Proceedings of the*

49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers, volume 2, pages 581–586, 2011.

Eran Hammer-Lahav. Rfc 5849: The oauth 1.0 protocol. *Internet Engineering Task Force (IETF)*, 2010.

Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 151–160, 2011.

Richard Johansson and Alessandro Moschitti. Relational features in fine-grained opinion analysis. *Computational Linguistics*, (Early Access): 1–37, 2012.

Sepandar D Kamvar and Jonathan Harris. We feel fine and searching the emotional web. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 117–126. ACM, 2011.

Jussi Karlgren, Gunnar Eriksson, Magnus Sahlgren, and Oscar Täckström. Between bags and trees–constructional patterns in text used for attitude identification. In *Advances in Information Retrieval*, pages 38–49. Springer, 2010.

Anders Kofod-Petersen. How to do a Structured Literature Review in computer science. Document released as a guide to write a Structured Literature Review at NTNU, 2012. URL `http://research.idi.ntnu.no/aimasters/files/SLR_HowTo.pdf`.

Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter Sentiment Analysis: The Good the Bad and the OMG! In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, pages 538–541, 2011.

Akshi Kumar and Teeja Mary Sebastian. Sentiment analysis on twitter. *IJCSI International Journal of Computer Science Issues*, 9(3):372–378, 2012.

Terje Nesbakken Lillegraven and Arnt Christian Wolden. Design of a Bayesian recommender system for tourists presenting a solution to the cold-start user problem. Master's thesis, Norwegian University of Science and Technology, 2010.

Kun-Lin Liu, Wu-Jun Li, and Minyi Guo. Emoticon Smoothed Language Models for Twitter Sentiment Analysis. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

Saif Mohammad and Peter Turney. Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 26–34. Association for Computational Linguistics, 2010.

Saif Mohammad and Tony Wenda Yang. Tracking sentiment in mail: how genders differ on emotional axes. In *Proceedings of the 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis (ACL-HLT 2011*, pages 70–79, 2011.

Saif Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 321–327, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S13-2053`.

Andrius Mudinas, Dell Zhang, and Mark Levene. Combining lexicon and learning based approaches for concept-level sentiment analysis. In

Proceedings of the First International Workshop on Issues of Sentiment Discovery and Opinion Mining, page 5. ACM, 2012.

Subhabrata Mukherjee, Akshat Malu, Balamurali AR, and Pushpak Bhattacharyya. TwiSent: A Multistage System for Analyzing Sentiment in Twitter. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2531–2534. ACM, 2012.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S13-2052`.

Finn Årup Nielsen. A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs. In *Proceedings of the 1st Workshop on Making Sense of Microposts (#MSM2011)*, pages 93–98, 2011.

Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

Alexander Pak and Patrick Paroubek. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In N. Calzolari, K. Choukri, and et al., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA). ISBN 2-9517408-6-7.

Bo Pang and Lillian Lee. *Opinion mining and sentiment analysis*, volume 2. Now Publishers Inc., 2008.

Sasa Petrovic, Miles Osborne, and Victor Lavrenko. The Edinburgh Twitter Corpus. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics in a World of Social Media*, pages 25–26, 2010.

Hassan Saif, Yulan He, and Harith Alani. Semantic Smoothing for Twitter Sentiment Analysis. In *Proceeding of the 10th International Semantic Web Conference (ISWC)*, 2011.

Hassan Saif, Yulan He, and Harith Alani. Semantic sentiment analysis of Twitter. In *The Semantic Web–ISWC 2012*, pages 508–524. Springer, 2012.

Beatrice Santorini. Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision). *Technical report MS-CIS-90-47*, 1990.

Øyvind Selmer, Mikael Brevik, Björn Gambäck, and Lars Bungum. NTNU: Domain Semi-Independent Short Message Sentiment Classification. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 430–437, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S13-2071`.

James Spencer and Gulden Uchyigit. Sentimentor: Sentiment Analysis of Twitter Data. In *SDAD 2012 The 1st International Workshop on Sentiment Discovery from Affective Data*, page 56, 2012.

Leonid Velikovich, Sasha Blair-Goldensohn, Kerry Hannan, and Ryan McDonald. The viability of web-derived polarity lexicons. In *Proceedings of the 2010 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 777–785. ACL, 2010.

WebProNews. Twitter Tops 500 Million Users; "Users" is a Loose Term, of Course, 2012. URL `http://www.webpronews.`

com/twitter-tops-500-million-users-users-is-a-loose-term-of-course-2012-07.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 347–354. Association for Computational Linguistics, 2005.

Ley Zhang, Riddhiman Ghosh, Mohamed Dekhil, Meichun Hsu, and Bing Liu. Combining lexicon-based and learning-based methods for Twitter sentiment analysis. *HP Laboratories, Technical Report HPL-2011-89*, 2011.

Appendix A

Systematic Literature Review Protocol

A.1 Introduction

This SLR protocol is developed during the specialization project of the fall semester 2012. This protocol will be used in both the specialization project and the master thesis.

Twitter is a microblogging platform used by millions of people all over the world. In contrast to other social media platforms, the Twitter messages, called tweets, are limited to a maximum length of 140 characters.

The goal of this project is to implement a bare-bone, modular and highly customizable application for doing sentiment analysis on tweets. In addition an extension of the existing Twitter API (Application Programming Interface) will be developed. This will be achieved by mimicking the API interface and passing on the query to the Twitter API. By simply extend-

ing the API, the sentiment analysis data will be easy to use for existing Twitter developers, and already be heavily documented by the Twitter API team.

The focus for this SLR is to search for papers with existing solutions for sentiment analysis on the Twitter corpus, in order to uncover the different performances and how the problem has been solved by other researchers.

A.2 Research Questions

RQ1 What are some of the existing solutions for SA (sentiment analysis) in the Twitter Corpus?

RQ2 How does the different solutions found by addressing RQ1 compare to each other with respect to micro-blogs like Twitter?

RQ3 What is the strength of the evidence in support of the different solutions?

RQ4 What implications will these findings have when creating the application/system?

A.3 Search Strategy

The domain used for the search will be Google Scholar. Google Scholar aggregates results from different domains, and has some built in functions for searching over synonyms and sorting by citations.

A set of terms is defined closely based on the first research questions (**RQ1**). The terms are split into groups where each group consists of words that are synonyms or have similar semantic meaning.

All search terms are placed in a table with the groups as columns and the search term as a row. The entire search string will be constructed by using

Boolean notation. All terms in a group are concatenated by the keyword OR, and the groups themselves are concatenated by the keyword AND. This search string is represented by the following formula:

([G1, T1] OR ([G1, T2] OR [G1, T3])
AND ([G2, T1] OR [G2, T2])

	Group 1	Group 2
Term 1	Sentiment Analysis	Twitter
Term 2	Sentiment Classification	Microblog
Term 3	Opinion Mining	

Table A.1: Search terms and groupings

All results found by using the search string, will be collected in a document, and reduced by removing duplicated papers, the same studies published from different sources and studies published before the year 2008.

A.4 Selection of Primary Studies

To reduce the studies even more, they are assessed using three different screenings; primary, secondary and by quality. The primary and secondary inclusion criteria are used to filter out the non-thematically relevant studies. The primary criterion is used on meta data such as title and abstract, while the secondary is used on the full text paper. The quality screening is also used on the full text as the last step of selection.

A.4.1 Primary Inclusion Criteria

IC1 The study's main concern is Sentiment Analysis.

IC2 The study is a primary study presenting empirical results.

IC3 The study focuses on sentiment analysis on the English language.

A.4.2 Secondary Inclusion Criteria

IC4 The study focuses on the Twitter corpus.

IC5 The study describes an implementation of an application.

All studies that make it through the primary and secondary selection criteria will be passed on to the quality assessments.

A.5 Study Quality Assessment

To further filter the papers and assess the quality of the different papers, a set 10 of quality criteria is defined. The first two criteria are used in quality screening, to assess whether the papers include basic research data.

Each study should be classified according to all 10 quality criteria. They can either be classified as "Yes" (1 point), "Partly" (1/2 point) or "No" (0 points).

QC1 Is there a clear statement of the aim of the research?

QC2 Is the study put into context of other studies and research?

QC3 Are system or algorithmic design decisions justified?

QC4 Is the test data set reproducible?

QC5 Is the study algorithm reproducible?

QC6 Is the experimental procedure thoroughly explained and reproducible?

QC7 Is it clearly stated in the study which other algorithms the study's algorithm(s) have been compared to?

QC8 Are the performance metrics used in the study explained and justified?

QC9 Are the test results thoroughly analysed?

QC10 Does the test evidence support the findings presented?

A.6 Data Extraction

From each paper, the following data will be extracted for the SLR:

- Study identifier
- Name of author(s)
- Title
- Year of publication
- Name of system
- Type of machine learning algorithm
- Data set source
- Findings and conclusions

The data will be presented in table format. Whereas the data type is divided into columns, and each paper is on its own row.