

Mateusz Drozdzyński

Sentiment Analysis for Multilingual Tweets

Computer Science Tripos

King's College

2011

Proforma

NAME	Mateusz Drozdzyński
COLLEGE	King's College
TITLE	Sentiment Analysis for Multilingual Tweets
EXAMINATION	Computer Science Tripos
YEAR	2011
APPROXIMATE WORD COUNT	8,100
PROJECT ORIGINATOR	Dr Daniele Quercia
PROJECT SUPERVISOR	Dr Daniele Quercia

Original Aims

Evaluate existing approaches for sentiment classification for languages other than English, focusing initially on four languages: German, Italian, Spanish and Polish.

Work Completed

The original aims were exceeded by evaluating the language and sentiment filtering assumptions in addition to carrying out the comparison of algorithms as per original aims.

Project Difficulties

Both Maximum Entropy and SVM classifiers had proven to be more difficult to implement than anticipated, but both have been incorporated with no or minor impact on project's success.

Declaration

I, Mateusz Drozdzyński of King's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date:

Contents

1	Introduction	1
1.1	Project Motivation	1
1.2	Aims	2
1.3	Related Work	2
1.3.1	Sentiment Analysis	2
1.3.2	Measuring Happiness	3
1.3.3	Crowdsourcing	4
1.3.4	Pattern Recognition and Machine Learning	4
1.3.5	Natural Language Processing	4
1.4	Glossary of Terms	4
2	Preparation	6
2.1	Research	6
2.2	Language Choice	6
2.3	Planning	7
2.4	Project Management	7
2.5	Tools Used	8
2.6	Data Sources	9
2.7	Algorithms	9
2.7.1	Feature Selection	10
2.7.2	Naïve Bayes	10
2.7.3	Maximum Entropy	10
2.7.4	Support Vector Machines	11
2.8	Available Tools	11
2.8.1	Naïve Bayes	12
2.8.2	Maximum Entropy	13
2.8.3	Support Vector Machines	13
2.9	Verification	14
2.10	Summary	14

3	Implementation	15
3.1	Overview	15
3.2	Infrastructure	15
3.2.1	External Dependencies	16
3.2.2	Storage	16
3.3	Twitter Crawler	17
3.3.1	Limitations	18
3.4	Feature Reduction	18
3.5	Testing and Evaluation	19
3.6	Classifiers	20
3.6.1	Baseline	20
3.6.2	Naïve Bayes	21
3.6.3	Maximum Entropy	22
3.6.4	Support Vector Machines	22
3.7	Verification	23
3.7.1	Results	25
3.8	Summary	26
4	Evaluation	27
4.1	Experimental Setup	27
4.1.1	Data Gathering	27
4.1.2	Ground Truth	27
4.1.3	Data Cleaning	30
4.1.4	Data Sets	30
4.2	Execution	30
4.2.1	Validation	30
4.2.2	Metrics	30
4.3	Results	31
4.3.1	English	31
4.3.2	Spanish	32
4.3.3	German	33
4.3.4	Italian	33
4.3.5	Polish	34
4.4	Discussion	34
4.5	Summary	35
5	Conclusion	36
5.1	Success	36
5.2	Challenges	36

5.3	Scope for Further Research	37
	References	38
A	Project Proposal	39
A.1	Introduction	40
A.2	Starting Point	40
A.3	Substance and Structure of the Project	40
A.4	Success Criteria	42
A.5	Plan of Work	43

List of Figures

1.1	Countries shaded by their position in the Happy Planet Index (2006), where highest-ranked countries are bright green and lowest are brown – Wikipedia	3
2.1	Illustration of an agile development process	8
2.2	Maximum-margin hyperplane and margins for an SVM trained with samples from two classes – Wikipedia	12
3.1	List of running and finished jobs – CrowdFlower	23
3.2	Visual job request editor – CrowdFlower	24
3.3	Survey interface for individual workers – CrowdFlower	25
3.4	Aggregated results for English – CrowdFlower	26

List of Tables

4.1	List of emoticons	28
4.2	Crowdfower	28
4.3	Language confidence	29
4.4	Sentiment confidence	29
4.5	Evaluation results for English tweets.	31
4.6	Evaluation results for Spanish tweets	32
4.7	Evaluation results for German tweets	33
4.8	Evaluation results for Italian tweets	33
4.9	Evaluation results for Polish tweets	34

Chapter 1

Introduction

1.1 Project Motivation

There is a growing interest in analysing and visualising data generated by users of social networks, which are expanding at an unprecedented rate. Business are looking into utilising these data streams in order to improve their marketing campaigns, refine advertising and better meet their customers needs. Governments of developed countries are also looking into ways of quantifying the wellbeing of their citizens in order to assess effectiveness of local governments and councils. Both tap into the social media in order to determine whether what people are saying is positive or negative. These applications introduce a problem of determining the sentiment of numerous yet short messages, often written in colloquial language. This makes it a very challenging research topic from the perspective of Natural Language Processing and Information Retrieval.

There have been attempts to tackle this problem, both in research and industry applications. Researchers have previously either focused on a different domain (e.g. film reviews as in Pang and Lee's paper [6]) or only on English (A. Go et al. [1]), which accounts for a mere 17.65% of the global population¹ and only about 50% of messages posted to Twitter every day.

My goal is to evaluate existing approaches for sentiment classification for languages other than English, focusing initially on four languages: German, Italian, Spanish and Polish. I shall also attempt to utilise crowdsourcing to verify the assumptions of robustness of language classification built into the Twitter API as well as the use of emoticons (e.g. smiley and frowny faces) as labels for training data.

¹http://en.wikipedia.org/wiki/List_of_countries_by_English-speaking_population

1.2 Aims

Evaluate and discuss the performance of sentiment classification techniques for English, German, Italian, Spanish and Polish tweets. The project shall attempt to achieve the following goals:

- Verify the assumption of reliability of language classification provided by Twitter’s API.
- Verify the assumption that presence of positive or negative emoticons can indeed be used to determine sentiment polarity.
- Explore differences in performance of selected algorithms for different languages.
- Attempt to explain potential sources of these differences.
- Highlight optimal sentiment classification techniques for each language.

1.3 Related Work

This project expands upon a number of previously published papers and articles, the summary of which is presented below.

1.3.1 Sentiment Analysis

Multiple papers have been published on sentiment analysis. Some have also explored using social media (and Twitter in particular) as their primary source of data.

Pang and Lee [6] present a comprehensive comparison of machine learning algorithms (Naïve Bayes, Maximum Entropy and Support Vector Machines) in a fairly narrow domain of film reviews. They evaluate all three algorithms in relation to a number of factors, such as use of unigrams, bigrams or parts of speech tagging. They conclude that whilst Naïve Bayes tends to offer inferior performance to Maximum Entropy and SVMs, the differences are typically negligible (1.9pp using unigram presence). They also determine that unigram presence is most effective model and none of the alternatives offered consistently superior performance.

A. Go et al. [1] apply the same techniques as Pang and Lee to Twitter messages, but focus solely on English. They also assume emoticons can be used effectively as noisy labels for positive and negative sentiment, quoting previous

research by J. Read [7], an assumption I verify independently in the Evaluation chapter.

1.3.2 Measuring Happiness

The term “Gross National Happiness” was coined by Bhutan’s former King Jigme Singye Wangchuck in 1972, when he tried to explain his commitment to aligning economic and cultural values of his country.

Since then there have been multiple attempts at mathematically modelling this concept in order to accurately measure and represent society’s wellbeing. Due to different factors being taken into account in these models, Gross National Happiness in that form is difficult to compare between multiple countries. Universal approaches, such as the Happy Planet Index (see Figure 1.1), offer a way to do so, but give little insight into how these values change with time (other than on yearly basis).

A. Kramer [4] explores the concept of computational Gross National Happiness index by analysing messages posted by approximately 100 million Facebook users and their activity on the platform. He uses a simplified bag-of-words model where GNH is represented as a difference between the number of positive and negative words in users’ status messages. He also validates that positivity of status messages does indeed reflect users’ life satisfaction scores. This result has important implications and justifies the need for performing similar analysis for languages other than English.

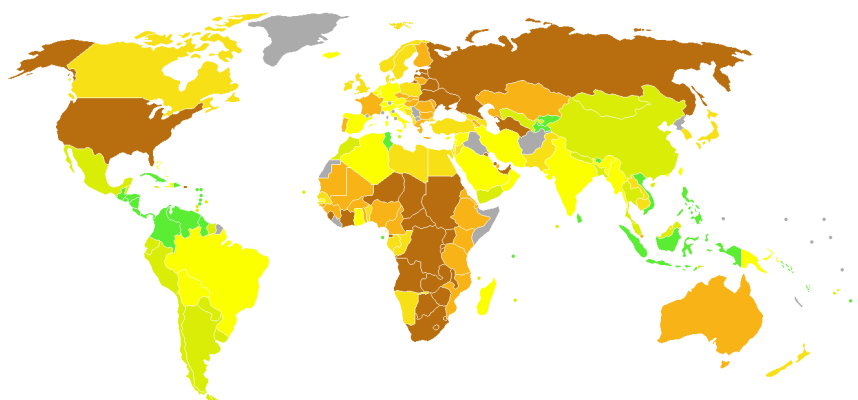


Figure 1.1: Countries shaded by their position in the Happy Planet Index (2006), where highest-ranked countries are bright green and lowest are brown – Wikipedia

1.3.3 Crowdsourcing

Understanding the limitations of crowdsourcing was considered an important part of this project, given that it was one of the few feasible solutions to verify some of the assumptions made about our data set.

A. Kittur, E. Chi and B. Suh [2] explain how crowdsourcing using micro-task markets such as Amazon’s Mechanical Turk² enables a low time and cost method for collecting user measurements. They offer a number of design recommendations which help achieve more accuracy and higher confidence scores from the use of such methods.

1.3.4 Pattern Recognition and Machine Learning

C. Bishop offers a comprehensive overview of Support Vector Machines and their applications as maximum margin classifiers. He discusses in detail how to model classification problems using linear models in a two-dimensional feature space.

His work has proven invaluable in understanding how Support Vector Machines work and how they can be applied to the problem of sentiment classification.

1.3.5 Natural Language Processing

P. Jackson and I. Moulinier elaborate on the differences between Bayesian information retrieval techniques (such as Naïve Bayes classification) and vector space techniques (such as SVMs). This publication provided an algorithmic background required to discuss these techniques in more detail in the Preparation section.

1.4 Glossary of Terms

This subsection explains and clarifies certain terms which are frequently used in subsequent parts of the dissertation.

- **Tweet.** A short message (limited to 140 characters) posted on the social networking and microblogging service called Twitter.
- **Sentiment.** A view of or attitude towards a situation or event; an opinion.
- **Emoticon.** A facial expression pictorially represented by punctuation and letters to alert the respondent to the attitude of statement’s author.

²<http://www.mturk.com>

- **Application Programming Interface (API).** An interface implemented by a software program to enable interaction with other software, similar to the way a user interface facilitates interaction between humans and computers
- **Gross National Happiness.** An abstract concept of quantifying happiness of citizens in a manner similar to how economy can be quantified using Gross National Product.
- **Crowdsourcing.** Practice of outsourcing tasks to an undefined, large group of people, primarily through the use of online tools.

Chapter 2

Preparation

This chapter explain preparation undertaken when analysing the requirement of the project as well as acquiring knowledge necessary to ensure timely completion of its goals. I also aim to explain some design decisions made along the way and their effect on the final outcome of this project.

2.1 Research

In my project I set out to explore existing tools for sentiment analysis and apply them to the problem of classifying multilingual tweets. It was clear from the onset of the project that I need to carefully choose which utilities or libraries to use as I was gearing towards exposing the results of my analysis through an API and a web application.

Research has therefore been made into not only existing toolkits such as WEKA, but also lower level libraries (in particular `libsvm` and `SVMlight`) which would later enable me to easily run the classifiers in the context of a web application.

2.2 Language Choice

I set out not only to evaluate existing solutions to sentiment classification across different languages but also lay foundations for a platform which can later be turned into a web application and an API. There are certain languages which are particularly well-suited for this purpose, such as Python, Ruby or PHP. All these languages have bindings to lower-level classification libraries, but Ruby in particular stood out by having a set of tools well-g geared towards building scalable

APIs, such as Rack¹ and Sinatra². Similarly hosting platforms such as Heroku³ make Ruby applications very easy to scale.

2.3 Planning

Planning and preparation are amongst factors significantly contributing towards a project's success. Moreover, little experience in Natural Language Processing (and machine learning techniques in particular) as well as the limited timescale of the project made it particularly ambitious.

I therefore needed to quickly verify my preconceptions and adopt to any unforeseen circumstances I might encounter along the way.

In order to do this efficiently, it was clear that I needed to build upon existing libraries and modify them only to the extent required by the scope of this project. Consideration also needed to be given to performance and scalability of the system, given it was planned to operate on a large data set in the future.

2.4 Project Management

In order to maintain flexibility of pursuing different paths in an area that is relatively new to me, I decided that a lightweight management approach such as Agile Software Development⁴ would be appropriate for this project, which has therefore been divided into a number of iterations, each containing individual goals that should be met before progressing onto the next one (see Figure 2.1). These are referred to in the Project Plan as 'Weeks' and allow for identifying problems with missing deadlines early on in the project.

I tried to follow many of the principles outlined by Andrew Hunt in his book, *The Pragmatic Programmer*[3], in which he offers an approach to software design which allows for developing and delivering high-quality software products.

Multiple factors were taken into account when establishing project milestones and their respective deadlines:

- Sufficient knowledge of Twitter API and a familiar development environment need to be established before any other work can be commenced.

¹<http://rack.rubyforge.org/>

²<http://www.sinatrarb.com/>

³<http://heroku.com>

⁴http://en.wikipedia.org/wiki/Agile_software_development

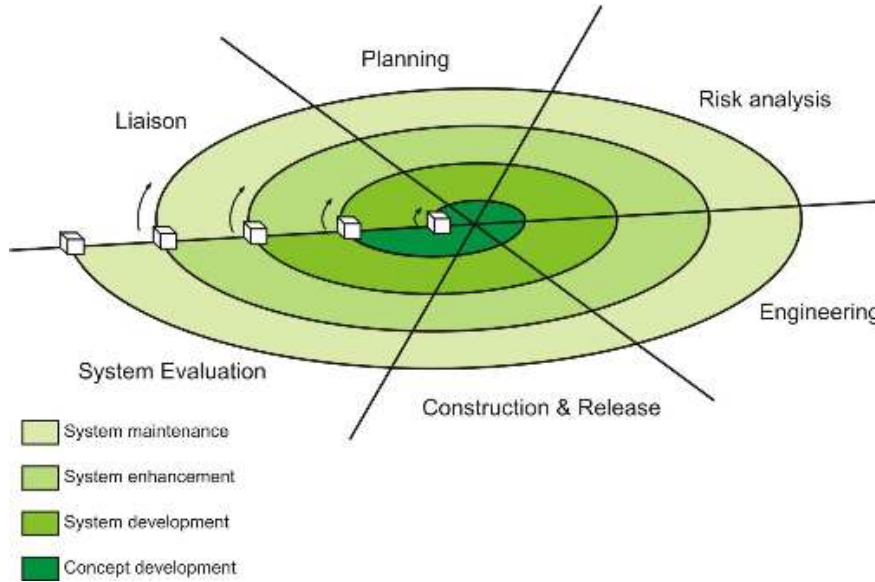


Figure 2.1: Illustration of an agile development process

- Preliminary data set needs to be gathered within first four weeks of the project in order to ensure evaluation work, even if limited, can be carried out.
- Evaluation and implementation chapters shall be written as subsequent classifiers are implemented.
- All classifiers shall be implemented before Easter Vacation.
- All chapters of the dissertation shall be written before the 2nd of May to allow for revision before exams.

This division was made possible by the modular nature of the project.

2.5 Tools Used

In order to avoid potential data loss and subsequent failure to meet the deadlines established in the project plan, a number of tools are used to ensure all code and the dissertation itself is properly versioned and backed up.

All code (including L^AT_EX source code for the dissertation itself) is stored in a version control system (git⁵). The repository is mirrored to GitHub⁶ and fur-

⁵<http://git-scm.com/>

⁶<http://github.com>

ther backups are stored on Dropbox⁷ and my personal Time Capsule⁸ to further mitigate any risk of data loss.

This provides a level of redundancy sufficient to finalise the project within desired timeline.

2.6 Data Sources

We use Twitter as our primary data source, primarily because it offers the biggest freely accessible source of real-time status updates. There are however some characteristics of Twitter messages that need to be taken into account when designing our experiments and evaluating their results.

- **Length.** Twitter messages are explicitly limited to 140 characters. Our dataset contains tweets which are 57 characters or 9 words long on average. It is interesting to note that the average length of a Twitter message seems to have fallen compared what has been shown in previous research [1] from 2009, where an average tweet was 78 characters (or 14 words) long. This can be attributed to sharing of links becoming a more prominent use case of Twitter.
- **Language model.** Twitter messages often contain slang, misspelling or even colloquially used phrases borrowed from different languages (English in particular).
- **Domain.** Twitter messages have a very diversified domain and are not centred about any particular topic.

Availability of tweets in different languages was also a concern. Due to both the limitations of the Twitter API as well as some languages being more likely to appear than the other, I needed to make provisions in case this process took longer than anticipated.

2.7 Algorithms

This section outlines mathematical models behind algorithms I used for sentiment classification.

⁷<http://dropbox.com>

⁸<http://www.apple.com/timecapsule/>

2.7.1 Feature Selection

Different approaches can be taken when selecting features to be fed into classification algorithms. Some of the libraries outlined in the Available Tools section offer built-in feature extraction capabilities (including Porter stemming).

As shown by A. Go et al. [1], there is little to be gained from going beyond the simplest unigram model. Therefore, for the purpose of this project we shall assume that each feature is a single word found in a tweet. Additionally, a number of feature reduction techniques described in the following chapter are employed in order to reduce the feature space.

2.7.2 Naïve Bayes

Naïve Bayes is a simple classifier based on applying Bayes' Theorem, often used for robust document classification due to its simplicity. One of its advantages is that it requires relatively small amount of training data before achieving peak performance.

In our model, a simple Bayesian classifier can be represented as a function mapping features $(f_1 \dots f_n)$ to a class (c) which is which this combination of features is most likely:

$$\text{classify}(f_1, \dots, f_n) = \operatorname{argmax}_c P(C = c) \prod_{i=1}^n P(F_i = f_i | C = c).$$

It is important to note that if we allow $P(F_i = f_i | C = c)$ to be equal to 0 (such as when our test vector contains features not present in the training data), the probability of that data point being in any class will also equal 0, irrespective of the likelihood of other features.

To avoid this phenomenon, we shall use *add-one* or *Laplace smoothing*, where we assume that all features, including unused ones, are initialised with a count of 1.

2.7.3 Maximum Entropy

Maximum Entropy is a group of feature-based models which do not assume independence of features, unlike Bayesian classifiers. In our simplified, unigram-based approach, this provides little benefit as features do not overlap. Additional complexity makes MaxEnt slower in comparison to Naïve Bayes, imposing additional limitations given our attempt to keep scalability in mind.

Maximum Entropy can also suffer from overfitting when training data is too sparse. In order to improve performance, we shall integrate a Gaussian prior by

using maximum a posteriori estimation instead of maximum likelihood. This is incorporated natively in the library that I use in this project.

2.7.4 Support Vector Machines

Support Vector Machines are a set of supervised learning methods which can be used as a non-probabilistic binary linear classifier by constructing a set of hyperplanes in a high or infinite dimensional space with the class assigned to new data points being dependent on their location in that space.

This model can be formalised by assuming we have a set of training data \mathcal{D} and a set of n points of the form

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$$

where y_i denotes the class to which point x_i belongs. We can now define any hyperplane as a set of points \mathbf{x} satisfying the following

$$\mathbf{w} \cdot \mathbf{x} - b = 0$$

where \mathbf{w} is a normal vector, where $\frac{b}{\|\mathbf{w}\|}$ determines the offset of given hyperplane along the normal vector \mathbf{w} .

Now in order to separate data into classes we define two further hyperplanes described by the equations

$$\mathbf{w} \cdot \mathbf{x} - b = -1$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = 1$$

and try to find \mathbf{w} and b to maximise the distance between parallel hyperplanes so that they are as far apart as possible whilst still separating the data. For an illustration see Figure 2.2.

2.8 Available Tools

This section explores existing tools for sentiment classifications and describes the rationale behind selecting those which had been used for this project.

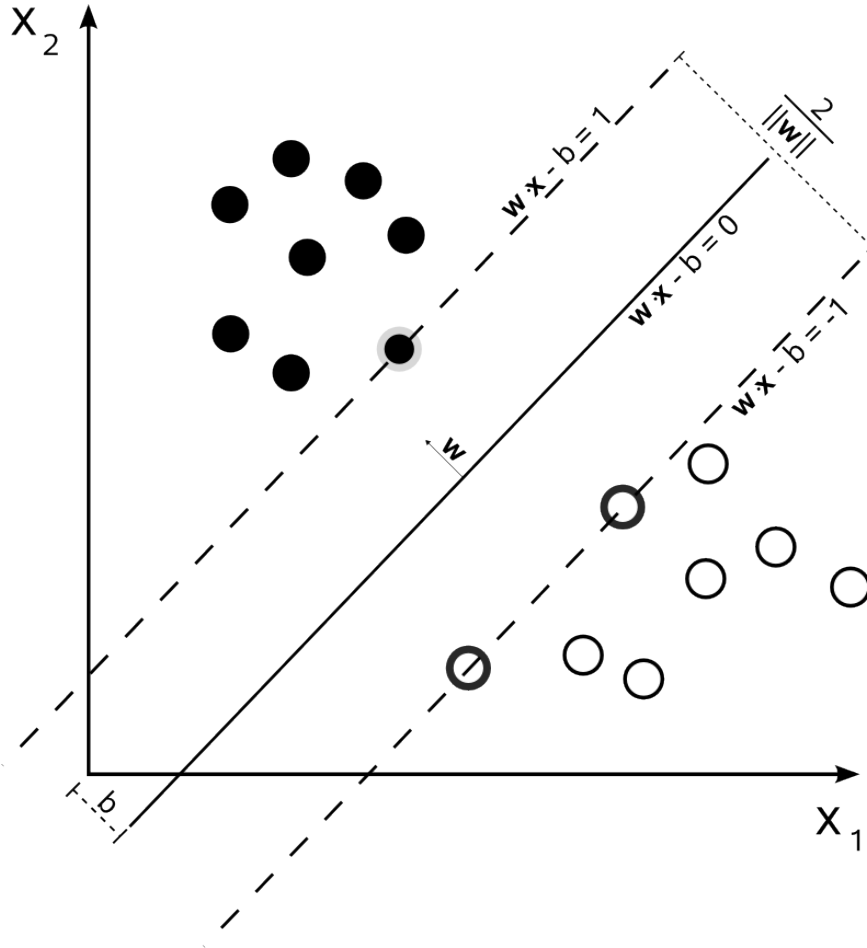


Figure 2.2: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes – Wikipedia

2.8.1 Naïve Bayes

The simplicity of Naïve Bayes classification means it is one of the most commonly implemented techniques in any machine learning package. It is therefore easy to find libraries that implement this particular classifier. Amongst the ones I explored were **WEKA**⁹, **Orange**¹⁰ and **Classifier4J**¹¹.

Both **WEKA** and **Orange** are GUI machine learning toolkits for data analysis and predictive modelling, particularly useful in research where there are few time and scalability concerns. I found them useful learning tools to experiment with

⁹<http://www.cs.waikato.ac.nz/ml/weka/>

¹⁰<http://orange.biolab.si/>

¹¹<http://classifier4j.sourceforge.net/>

different classification methods, but in order to meet the requirements of this project, a scalable lower-level solution was needed.

`Classifier4J` meets most of these criteria, but due to limited documentation and lack of activity in the project since 2005 I abandoned it in favour of a more modern solution.

In the end I settled on a Ruby library `Ankusa`, which due to its support for Hadoop's `HBase`¹² and `Cassandra`¹³ for storage offered promising scaling capabilities. My familiarity with Ruby has also proven helpful in understanding the internals of the library and enabled me to modify it to the extent required by this project.

2.8.2 Maximum Entropy

Unfortunately there are very few Ruby libraries that implement Maximum Entropy classifiers. Initially, attempts were made to use `maxent`¹⁴, which had proven difficult given lack of documentation and updates since 2008 (since then Ruby 1.9.x became standard and many 1.8.x libraries stopped working).

In the end I decided to use `maxent_string_classifier`¹⁵, which is based on the `OpenNLP`¹⁶ `Maxent` framework. The library is well-maintained and has reasonably comprehensive documentation. The only disadvantage of using `maxent_string_classifier` is that it introduces a dependency on `JRuby`¹⁷, which is a Java implementation of the Ruby programming language.

2.8.3 Support Vector Machines

Having been warned on multiple occasions that Support Vector Machines might be difficult to integrate, I decided to research them ahead of schedule to avoid any potential delays. I looked into GUI tools such as `WEKA`, but none of them have the scaling properties required to parse a large number of messages in real time.

After thorough considerations, `libsvm`¹⁸ and `Eluka`¹⁹ were chosen as they seemed to meet my performance requirements whilst remaining relatively easy to use.

¹²<http://hbase.apache.org/>

¹³<http://cassandra.apache.org/>

¹⁴<http://mastarpj.nict.go.jp/~mutiyama/software.html#maxent>

¹⁵https://github.com/mccraigmccraig/maxent_string_classifier

¹⁶<http://incubator.apache.org/opennlp/>

¹⁷<http://www.jruby.org/>

¹⁸<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

¹⁹<https://github.com/arrac/eluka>

2.9 Verification

This project makes a number of assumptions regarding the training data set that needed to be verified manually in order to establish confidence of results presented. Firstly, we assumed that Twitter’s built-in language filter works reliably, for which there was no evidence by either Twitter or any third parties. Secondly, it is difficult to determine the actual sentiment of a tweet because of humour, sarcasm and non-standard use of emoticons, potentially rendering our assumption that they can be used as labels for training data unfounded.

Since manual verification would require access to a number of people fluent in five different languages evaluated in this project, I first attempted to find a solution which would make this process easier.

In my preliminary research, Amazon Mechanical Turk²⁰ seemed to be the forerunner in providing on-demand, scalable workforce for simple tasks such as manual classification of data. This solution was however quickly discarded as becoming a requester on the platform requires a US billing address.

I then shifted my focused to companies and tools which act as intermediaries between researches and Amazon’s Mechanical Turk. This lead me to exploring and subsequently settling on CrowdFlower²¹, which allows for publishing jobs not only to AMT but also other crowdsourcing platforms. It also offers an internal interface should any of the providers fail to complete requested tasks.

2.10 Summary

In this section I described the research undertaken in order to successfully plan the project and gather information about various machine learning techniques and their applications in the area of sentiment classification. I also discussed how my assumptions regarding the use of emoticons and Twitter’s built-in language filter will be verified.

²⁰<https://www.mturk.com/>

²¹<http://crowdflower.com/>

Chapter 3

Implementation

This chapter aim to elaborate on the implementation details of the project undertaken in order to establish a platform which will allow for evaluating different classification algorithms and subsequently drawing conclusions from such evaluation.

3.1 Overview

The system consists of the number of independent components, further described in subsequent sections. Each of this components is expected to provide a certain set of methods, later referred to as APIs. As long as these requirements are met, those modules can easily be replaced or modified provided they adhere to the same interface.

This modularity enabled me to work concurrently on different parts of the system without any significant interference. It also contributed to the scalability and parallelisation of the system, where crawlers and classifiers could easily be distributed across different servers and still perform adequately.

The system is split into three primary components: core infrastructure for handling Twitter messages, database connections and scores; a crawler which was used to create the data set and a set of classifiers with a generic runner. The architecture of these components is explained in more detail in the following sections.

3.2 Infrastructure

In order to avoid code repetition and discrepancies between different classification methods, I implement a set of common interfaces which are then used by

higher-level implementations in order to facilitate access to the data set, feature reduction algorithms, running the classification algorithms and computing performance scores.

The system is kept modular by isolating commonly used abstractions in the `common` folder. It contains implementations of `Score` and `Tweet` classes, which provide interfaces for working with individual Twitter messages and cumulative classification scores.

The `Tweet` class assumes access to the data set through a set of APIs provided by `ActiveRecord`. This allows for immediate support for multiple relational data stores. Through the use of `ActiveModel` this support can be extended to any other data source, such as `Hadoop`¹ or even plain text files, as long as they adhere to a predefined interface.

3.2.1 External Dependencies

I use `Bundler`² to manage application's dependencies across multiple machines. Having specified these dependencies, `Bundler` will keep track of required gems and their version numbers in order to establish a consistent execution environment, regardless of the machine at which the application is run.

3.2.2 Storage

An SQLite database was used to store both training and test data. Working with a relational database rather than files allowed for easier formatting and querying the data, especially with respect to filtering out messages of certain language or sentiment.

Messages are stored in a single `tweets` table with the following columns: `status_id`, `language`, `text` and `json`, where:

- `status_id` is a unique Twitter status identifier
- `language` is an ISO 639-1 coded language of the message
- `text` is the full text of the status message
- `json` is the original JSON response as received from the Twitter API.

The original JSON response is retained so that parameters other than `status_id`, `language` or `text` could be recovered if necessary. I de-normalise

¹<http://hadoop.apache.org/>

²<http://gembundler.com/>

the schema and introduce indexes to be able to efficiently filter out records by any of aforementioned parameters. Consistency is maintained because records are not modified after they have been added to the database.

3.3 Twitter Crawler

Before any analysis could be carried out it was necessary to gather enough training and test data.

For this purpose, I built a simple crawler which queried Twitter Search API³ and persisted the results in a local database for further processing and analysis.

In order to avoid having to interact with Twitter API⁴ directly, I used an open source Twitter gem⁵ which implements OAuth authentication as well as easy API access in Ruby⁶.

There are two features of the Twitter API that are crucial to the operation of our crawler:

- **Language filter.** Passing an optional `lang` parameter restricts tweets to a given language, given by an ISO 639-1 code.
- **Attitude filter.** Querying for `:)` and `:(` returns tweets with positive and negative attitude respectively. This feature is not documented in the Search API, but has been announced⁷ by Twitter and is listed amongst other search operators⁸.

The crawler will therefore carry out a separate query for each sentiment class in each language. For example, when retrieving positive tweets in English, the wrapper would send a HTTP GET request to `http://search.twitter.com/search.json?q=:)&lang=en`.

The result of such request is then parsed and each message is instantiated as a `Tweet` object, later persisted by invoking the `save` method provided by `ActiveRecord`. A utility method `Tweet#parse` was written to make this process simpler.

The crawler was then run in the cloud using Amazon Elastic Cloud Computing⁹ service (EC2), part of the Amazon Web Services infrastructure. Running

³<http://dev.twitter.com/doc/get/search>

⁴<http://dev.twitter.com/doc>

⁵Gems are self-contained packages for distributing Ruby programs and libraries.

⁶<https://github.com/jnunemaker/twitter/>

⁷<http://twitter.com/twitter/status/2262419764>

⁸<http://search.twitter.com/operators>

⁹<http://aws.amazon.com/ec2/>

the crawler on a remote server instead of on my local machine mitigated the risks associated with both connection and hardware failures. The database was regularly downloaded and backed up to my personal Dropbox account to avoid potential data loss.

3.3.1 Limitations

Since results filtered with the `lang` parameter are limited to last 7 days and the API itself imposes a rate limit, I needed to run the crawler over a few weeks with idle intervals of 10 minutes between each run (which involves 10 different queries, two for each of the five languages) to avoid exceeding said limit. Since Twitter reserves the right to block applications which do not comply to its terms of service, it was critically important that these limits are respected in order to successfully complete the project.

3.4 Feature Reduction

There are several properties, some unique to Twitter, that can be taken advantage of in order to reduce the feature space.

- **Case sensitivity.** In order to avoid differentiating between spelling variations of the same word (e.g. `Love`, `love` or even `lOvE`), all documents are converted to lower case first.
- **Usernames.** Since Twitter users often direct messages to each other by prepending other people's Twitter handles with an `@` symbol, I introduce the `USERNAME` equivalence class token, replacing all words that start with the `@` symbol.
- **Links.** Similarly, Twitter users often include links in their messages. I therefore introduce another equivalence class which maps all URLs to the `URL` token.
- **Repeated letters.** Due to the prominent use of casual language on Twitter, repeated letters are often used as a form of emphasis (e.g. writing `loooooooooove` to mean `love`). I therefore ignore all duplicate letters after the first two occurrences (i.e. mapping `loooooooooove` to `loove`).

This process, also known as feature extraction, is important since classification can be done more accurately in a reduced rather than original space.

All these reductions are performed on-the-fly using regular expressions, the list of which is defined in the `Tweet#reductions` method so that it could be easily extended if necessary.

3.5 Testing and Evaluation

In order to maintain the consistency of testing between different classification methods, the `Classifier::Base` class provides a unified way of performing and evaluating classifications using k-fold cross validation. By default a 10-fold cross-validation is used, but the library allows to change the number of folds if necessary.

Firstly, the crawler queries the database to collect two data sets for each language—one containing only positive and one only negative tweets. It also initiates a `Score` instance. New data points can then added using the `Score#add` method, which compares sentiment passed on from the classifier with tweet’s original label and increments appropriate counters. These counters can then be used to calculate precision and recall for individual folds as well as an average cumulative result.

I use the following definitions when computing aforementioned metrics:

- $\text{precision}(c) = \frac{\# \text{ of documents correctly classified as 'c'}}{\text{total } \# \text{ of documents classified as 'c'}}$
- $\text{recall}(c) = \frac{\# \text{ of documents correctly classified as 'c'}}{\text{total } \# \text{ of documents}}$

Where c in the equations above stands for the sentiment class, i.e. `positive` or `negative`.

`Score` class also implements two helper functions: `summary` and `latex`. The former outputs the results held in the `Score` instance in a human-readable form: stating which classifier was used, which language is being evaluated as well as individual and cumulative performance metrics. The latter outputs the same results in the L^AT_EX format, which can be directly embedded in documents such as this dissertation.

Instances of the `Score` class can also be easily added together with the `Score#+` method, which overrides the default addition operator. The new score inherits the label, classifier and language of the left-most operand and merges all internal counters.

Individual implementations, such as `Classifier::NaiveBayes`, need only to extend the `Classifier::Base` class to inherit most of its functionality, whilst implementing their own classification techniques. This can be achieved by calling the

`evaluate` method and passing a block of code containing the classification procedure. Test and training data along with the `Score` object are passed as arguments to this block and do not have to be instantiated in each implementation. Each of these classes can change the default parameters set by `Classifier::Base`, including the list of languages to be evaluated, the number of tweets per language or the number of folds performed.

3.6 Classifiers

This section outlines the implementation of various sentiment classification techniques, including the machine learning techniques to be evaluated. Since most classifiers have been implemented using readily-available libraries, it is primarily concerned with changes that needed to be made in order to use these libraries in the project, includes normalising input/output differences between algorithms and additional parameters that might affect evaluation consistency.

3.6.1 Baseline

As a baseline, I decided to use a keyword-based dictionary count with a list of 405 positive and 500 negative keywords¹⁰, an approach that has been successfully used by Pang and Lee [6]. Some words in the dictionary are represented using regular expressions and thus entries such as `'admir\w*'` will match multiple words (e.g. `'admiration'` and `'admiring'`).

The classifier is implemented as follows:

For each tweet, separate polarity counts for both sentiment classes are calculated and the one with a higher score is returned. Should there be a tie, the classifier will return the result as `'unclassified'`. We define polarity as the number of words in the message present in the respective dictionary, therefore only presence, not frequency, is taken into account. Similarly we do not express the polarity as a function of document length as its variance in our corpus is relatively small.

Translation

Since annotated dictionaries for languages other than English are difficult to obtain, I evaluated different approaches in order to establish a baseline for languages other than English.

¹⁰<http://www.liwc.net/>

- **Dictionary translation.** In this approach we first use a dictionary to create a list of all positive and negative English words captured by the keyword-based dictionary. For example, an ‘`anqr\w*`’ entry would be expanded into ‘`anqrily`’, ‘`anqriness`’, ‘`anqrte`’ and ‘`anqr`’. This expanded list of keywords is then fed into Google Translate and translated into Spanish, Italian, German and Polish.
- **Document translation.** In this approach, instead of translating the dictionary, we translate individual tweets into English and use the exact same baseline algorithm for all documents, regardless of their original language.

Both approaches have certain advantages and disadvantages. Translating the dictionary might result in slightly lower recall, as certain words might not be generated in the translation process. It is however a robust approach which doesn’t require post-processing a large data set of tweets. Moreover, freely available translation tools, such as Google Translate, have certain limitations which make them unsuitable for translating large numbers of messages on the fly. In this project I therefore only explore the first approach, i.e. dictionary translation. In order to obtain comparable results for English and rest of evaluated languages, an expanded version of the dictionary (2425 positive and 3071 negative words with no wildcards) was used for all languages, including English.

3.6.2 Naïve Bayes

Ankusa¹¹ is a text classification library in Ruby which provides Naïve Bayes and Kullback-Leibler divergence classifiers and utilises Hadoop’s **HBase** and **Cassandra** for storage, therefore providing a scalable solution for corpuses of many terabytes in size.

The scope of this project only covers the use of the Naïve Bayes classifier and does not use either of the distributed stores for simplicity. Memory and single file storage has been proven sufficient for corpora of the size used in this dissertation.

The library also provides two additional features that can significantly affect classification accuracy, and therefore need to be discussed in relation to the remaining algorithms:

Stop words. Ankusa automatically filters out a number of stop words. Since the library only provides a dictionary of common English words, it may introduce a discrepancy in classification accuracy between languages. For the purposes of this project I therefore disabled this feature in order to establish consistent results.

¹¹<https://github.com/livingsocial/ankusa>

Stemming. Ankusa uses the Porter stemming algorithm for the remaining words in the document, a feature also only available for English. Similarly, it needed to be disabled for the reasons listed above.

Upon removal of these features we can reliably compare performance of this classifier for different languages. In order to evaluate them independently, we initiate new `Ankusa::MemoryStorage` and `Ankusa::NaiveBayesClassifier` and re-train the classifier in each fold. We can then use the `classify` method of `Ankusa::NaiveBayesClassifier`, which returns the most likely class for given document, to determine whether classification was successful.

3.6.3 Maximum Entropy

Finding a library which offered robust support for Maximum Entropy classification was one of the biggest challenges of the project, as few of the solutions met the requirements outlined in the previous chapters. Having said that, `maxent_string_classifier` turned out to be fairly easy to integrate despite certain differences between Naïve Bayes and SVM libraries.

`maxent_string_classifier` does not offer support for loading training data form within Ruby and requires it is put into files. Our `Classifier::MaxEnt` class was therefore made responsible for exporting records from the database into text files and instructing `MaxentStringClassifier::Loader` to use them as training data. From then on, classifications can be made through the `classify` method similarly to other libraries.

3.6.4 Support Vector Machines

`Eluka`¹² is a `libsvm` based Support Vector Machine classifier for Ruby. It also provides an interface to `fselect.py`, a feature selection library written in Python.

Due to the experimental nature of the library, some amendments needed to be made before it started producing consistent results.

Firstly, the library tries to establish a single class for all test data points. In order to avoid having to rebuild the model after classifying each data point, I implemented a `reset` method which re-initiates the test feature vector instead. This ensures that all documents in the test corpus are classified independently.

`Eluka` also uses `fselect.py` for feature selection should one pass a string to the `add` method of the classifier. Since we want to have control over how the feature vector is generated in order to established results consistent with other

¹²<https://github.com/arrac/eluka>

classifiers, we manually generate a `Hash` object representing the feature vector and pass it on to the library.

The same method is then used to generate the feature vector in the test protocol.

3.7 Verification

In order to verify the assumption positive emoticons indeed indicate positivity of the message (similarly for negative emoticons and negativity of the message), I use a crowd-sourcing service CrowdFlower¹³ to manually labels a subset of my data set for further analysis and verification.

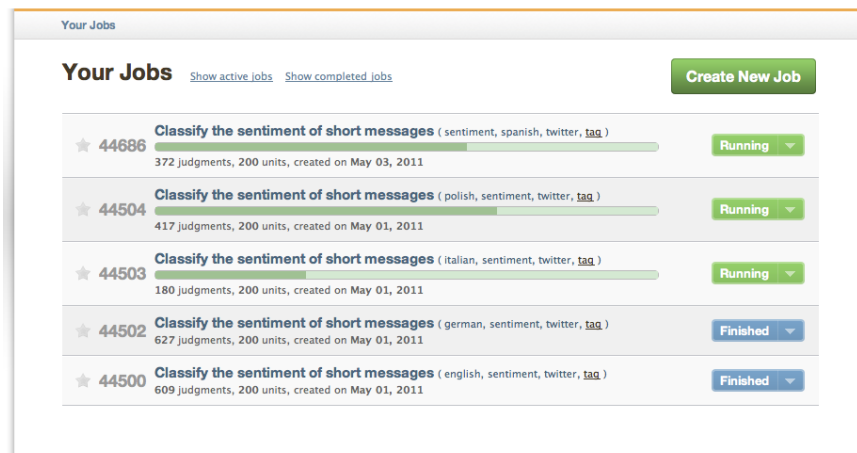


Figure 3.1: List of running and finished jobs – CrowdFlower

In order to generate sample data, a new `Classifier::Crowdfower` class was introduced, which shares the basic infrastructure with the rest of the classifiers, with the exception of the evaluation, which was handled manually. This class outputs several CSV¹⁴ files, in the format which can then be imported into Crowdflower. Each file contains 200 messages, equally divided between both polarities, for a total of 5 files containing 1,000 tweets selected for manual verification.

CrowdFlower provides an online editor (see Figure 3.2) which allows to create and test tasks before they are distributed to remote workers. Since I was trying to verify two separate assumptions (language and sentiment filter confidence),

¹³<http://crowdfower.com>

¹⁴Comma-separated values

users were presented with two questions per tweet, along with a note describing the task and clarifying individual questions as necessary:

Classify the sentiment of sort messages. Please read the messages below and, for each of them, confirm which language it is written in and determine whether the attitude of the speaker is positive, negative or neutral.

1. **Language:** English, Spanish, German, Italian, Polish.
If the message is written in more than one language, please try to determine which is the primary language of the message. E.g. ‘I love fußball’ shall be considered English, even though it contains German words. Similarly ‘Ich liebe football’ would be considered German.
2. **Sentiment:** Neutral, Positive, Negative.

Title & Description

This is the first thing the workers will see. Be clear and concise, so that workers can find task easily.

Helpful Tips

- The title should quickly summarize exactly what the worker would be doing if they accepted your job.
- The description should contain the instructions your workers will need to successfully complete this task. Be as clear and comprehensive as possible. If you think something might confuse some workers, clarify here.
- Resist the temptation to overload your title and description with spammy keywords like "EASY!", "fast", "\$\$\$", etc. If you inaccurately describe your job, workers will not want to work on it.

Example: "Flag images that contain adult content"

Classify the sentiment of short messages

Please read the messages below and, for each of them, confirm which language it is written in and determine whether the attitude of the speaker is positive, negative or neutral.

Rich text editor toolbar

{{ text }}

Language (required)
 English

If the message is written in more than one language, please try to determine which is the primary language of the message. E.g. "I love fußball" shall be considered English even though it contains German words. Similarly "Ich liebe football" would be considered German.

Sentiment (required)
 Neutral

Figure 3.2: Visual job request editor – CrowdFlower

Certain location requirements (see table below) were also imposed on workers in order to minimise the risk of them not speaking the language in which the message they are evaluating was written. Each job was also tagged with certain keywords (**language**, **sentiment**, **twitter**, **attitude** and **tweet**) which help match workers to tasks they feel comfortable doing.

Language	Location
English	United Kingdom
Spanish	Spain
German	Germany
Italian	Italy
Polish	Poland

Ordered jobs are then distributed to workers in selected countries using four different delivery channels (Amazon’s Mechanical Turk¹⁵, Gambit¹⁶, Give Work¹⁷ and CrowdFlower’s internal interface). Each worker is then presented with an interface allowing them to make up to 50 judgements as illustrated in Figure 3.3.

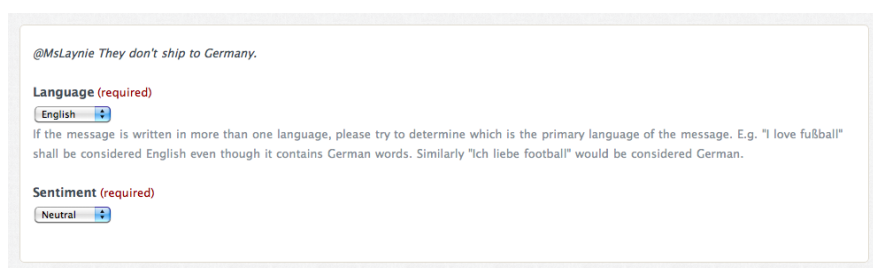


Figure 3.3: Survey interface for individual workers – CrowdFlower

Workers are remunerated depending on the number of judgements they make for a predefined price per task. Evaluation of our sample of 1,000 units and 3,000 judgements costs under \$60, making CrowdFlower an appealing alternative to carrying out surveys, which would take hours of researchers’ time. However, despite CrowdFlower’s focus on large-scale data, evaluating a relatively small sample took over two weeks. This level of performance was acceptable in the scope of this project, but might not be for someone trying to replicate the results in a shorter timeframe.

3.7.1 Results

CrowdFlower delivers the results in two different formats—full and aggregated—both in form of a CSV spreadsheet (see Figure 3.4). In case of the former, each row represents an individual judgement made by a worker for a given tweet. Since individual tweet will have multiple (at least 3) judgements, I prefer to use

¹⁵<http://www.mturk.com>

¹⁶<http://www.getgambit.com/>

¹⁷<http://www.samasource.org/iphone>

the aggregated summary where multiple judgements are collapsed down and a confidence score is given to represent workers' agreement on the sentiment and language of each tweet.

_unit_id	_trusted_judgm	_last_judgment	language	language:conf	sentiment	sentiment:conf	orig_language	orig_sentiment	status_id	text
72512024	3	5 Mar 2011 11:00	English	1.0	Positive	1.0	en	positive	5.95594E+16	@Eleanordann i
72512025	3	5 Jan 2011 21:00	English	1.0	Positive	1.0	en	positive	5.70239E+16	Great Night
72512026	3	5 Jan 2011 11:00	English	1.0	Positive	1.0	en	positive	5.84347E+16	@bec_juvsJB th
72512027	3	5 May 2011 21:00	English	1.0	Neutral	0.6592	en	positive	4.91574E+16	10 more follow
72512028	3	5 Apr 2011 21:00	English	1.0	Positive	0.6879	en	positive	5.84245E+16	@MisticeFleur t
72512029	3	5 Aug 2011 11:00	English	1.0	Neutral	0.676	en	positive	6.05922E+16	@DrewParks Ar
72512030	4	5 Aug 2011 21:00	Other	0.5097	Neutral	1.0	en	positive	5.73029E+16	@anssp cieeee
72512031	3	5 Apr 2011 11:00	English	1.0	Neutral	0.6879	en	positive	5.70239E+16	@skymptofAngel
72512032	3	5 Aug 2011 11:00	English	1.0	Neutral	1.0	en	positive	5.94956E+16	@GottentGonD
72512033	4	5 Aug 2011 21:00	English	1.0	Neutral	0.7269	en	positive	5.04228E+16	Watching TV w/
72512034	3	5 Apr 2011 21:00	English	1.0	Negative	0.6893	en	positive	5.14842E+16	At BW's and he
72512035	4	5 Aug 2011 21:00	English	1.0	Neutral	0.7259	en	positive	5.00647E+16	@BrandonPDur
72512036	3	5 Mar 2011 11:00	English	1.0	Positive	0.6879	en	positive	6.05079E+16	Candy cabs lov
72512037	3	5 Aug 2011 11:00	English	1.0	Positive	1.0	en	positive	6.02392E+16	@JdeAnacker45
72512038	4	5 Aug 2011 21:00	English	1.0	Neutral	0.7628	en	positive	5.97179E+16	Mere Saem Ch
72512039	4	5 Aug 2011 21:00	English	1.0	Neutral	0.5013	en	positive	5.0525E+16	@Kirsteencccc
72512040	3	5 Jan 2011 21:00	English	1.0	Positive	1.0	en	positive	5.77168E+16	20 more sleeps
72512041	3	5 Apr 2011 11:00	English	1.0	Positive	1.0	en	positive	5.87209E+16	feeling pretty hi
72512042	3	5 Mar 2011 11:00	English	1.0	Neutral	0.6596	en	positive	6.22256E+16	Tiramisu, straw

Figure 3.4: Aggregated results for English – CrowdFlower

Calculations normally handled by `Classifier::Base` through the `Score` class have been carried out using desktop spreadsheet software, in this case Apple Numbers (part of the iWork¹⁸ productivity suite), in order to avoid having to import augmented CSV files back into the project.

3.8 Summary

In this chapter I described how the infrastructure for evaluating different classifiers was implemented, how test and training data was collected and how the assumptions about the data set were evaluated. The latter is an extension which had not been part of the original proposal and was completed instead of focusing on implementing a sample application or an API, as suggested in the Overseers' Report. This provided invaluable insight into the quality of data gathered and how it relates to the overall performance of evaluated algorithms.

¹⁸<http://www.apple.com/iwork/>

Chapter 4

Evaluation

In this chapter I present the results of accuracy tests for implemented classification methods. I also describe the experimental setup and discuss any trade-offs involved. Given that evaluation is a substantial part of the project, special care was taken in designing the testing protocol and ensuring all assumptions have been verified and tested before reaching any conclusions.

4.1 Experimental Setup

4.1.1 Data Gathering

I had access to a public dataset of Twitter messages in English with their associated sentiment to begin with. This however does not extend to other languages, so collecting my own data was necessary. This was achieved using Twitter’s Application Programming Interface (API), for which I had written a simple scraper in Ruby.

The API has a built-in language filter which I have used to query for Tweets in specific languages involved in my experiment. It has to be noted that this method of discerning between different languages is not impeccable, especially considering how English loan-words can often be used.

4.1.2 Ground Truth

It has been shown by J. Read [7] that emoticons can be used as noisy labels to perform distant supervised learning. We further tested this assumption with the aid of crowd-sourcing. Since multiple emoticons can be associated with either of the two sentiment classes, we use a mapping which can be found in Table 4.1.

Emoticons mapped to :)	Emoticons mapped to :(
:)	:(
:-)	:-(
:)	: (
=)	

Table 4.1: List of emoticons

Queries are handled natively by Twitter API, which when asked for messages containing ':')' will return all tweets which contain positive emoticons and conversely for ':('.

It is also assumed that the language of the message is correctly determined by Twitter.

Verification

I verify my assumptions regarding sentiment and language filtering by manually classifying both properties for a randomly selected subset of tweets.

Language	# of tweets	# of judgements
English	200	609
Spanish	171	368
German	200	627
Italian	71	143
Polish	130	399

Table 4.2: Crowdfower

In order to analyse the results let me first define the following terms:

- **Agreement.** A judgement in which the worker agreed with the original classification label.
- **Disagreement.** A judgement in which the worker disagreed with the original classification label.
- **Confidence.** Ratio of the number of judgements agreeing on the language or sentiment of a tweet over the total number of judgements.
- **Accuracy.** The likelihood that a single tweet has been correctly classified.

Language	Agreements		Disagreements		Accuracy
	#	Confidence	#	Confidence	
English	193	99.40%	7	76.23%	96.75%
Spanish	152	99.52%	19	82.22%	90.44%
German	70	96.12%	130	83.79%	44.19%
Italian	13	93.42%	58	92.36%	23.35%
Polish	25	97.32%	105	90.08%	26.73%

Table 4.3: Language confidence

High confidence in both agreements and disagreements with the original labelling implies that the workers can indeed be trusted and do not differ significantly in their opinions on tweets’ language (see Table 4.2). This is a reassuring verification of the scientific method. However, the accuracy of Twitter’s native language filtering is somewhat worrying. It appears that whilst it can identify English and Spanish with 90%+ accuracy, it performs substantially worse for other tested languages I tested. Workers reported that only 23% of messages labeled by Twitter as Italian were indeed Italian.

This introduces a need for significant improvement over Twitter’s native language classification, something that I would like to explore in future research.

Language	Sentiment	Agreements		Disagreements		Accuracy
		#	Confidence	#	Confidence	
English	+	56	88.35%	54	79.87%	54.86%
	-	44	83.27%	56	73.45%	51.51%
Spanish	+	54	82.48%	46	84.39%	51.72%
	-	30	100%	41	97.56%	40.69%
German	+	35	80.76%	65	85.44%	37.73%
	-	33	79.86%	67	85.87%	35.82%
Italian	+	13	87.15%	36	88.89%	31.29%
	-	9	100.00%	13	96.87%	50.65%
Polish	+	38	87.00%	62	84.40%	42.73%
	-	7	70.61%	30	80.85%	28.89%

Table 4.4: Sentiment confidence

The assumption that emoticons can be used reliably as a way of discerning between positive and negative messages has also been questioned by the results (see Table 4.4). It appears that whilst the disparity between languages is much

smaller than in case of language classification, the performance is still subpar with accuracy between 28.89 and 54.86%.

With this level of error introduced by both of these assumptions it is easy to question relative differences in performance between classifiers.

4.1.3 Data Cleaning

For the purpose of training, the following post-processing filters are used:

1. Emoticons are removed as otherwise they would affect accuracy as shown in previous research [1].
2. Repeated tweets are removed so that each tweet is considered only once.
3. Retweets are removed as they are in principle just repeated tweets. Retweeting is a process of taking a Twitter message someone else had posted and rebroadcasting it to your followers. This is done by ignoring all messages containing the 'RT' keyword. A different keyword—'via'—can also be used, but less so for verbatim retweets [5], hence the decision not to filter it out.

4.1.4 Data Sets

We use data sets of equal cardinality in order to control for differences in performance of machine learning algorithms in relation to training data size. Data set for each language contains 150,000 tweets (75,000 positive and 75,000 negative).

4.2 Execution

4.2.1 Validation

Naive Bayes, Maximum Entropy and Support Vector Machines experiments are performed using 10-fold cross validation.

We partition our sample randomly into 10 subsamples, retaining one as validation data for testing and use the remaining 9 subsamples for training. We repeat this process 10 times using different subsamples as validation data.

In each trial we compute and record metrics explained below.

4.2.2 Metrics

To test the three algorithms I measure precision, recall and f-score (as defined below) over the entire dataset as well as individual classes.

We define precision and recall as follows:

$$\text{precision} = \frac{\text{number of correctly classified tweets}}{\text{total number of classified tweets}}$$

$$\text{recall} = \frac{\text{number of correctly classified tweets}}{\text{total number of tweets}}$$

Subsequently I average the results of all trials to arrive at the final evaluation metric for each algorithm.

4.3 Results

The results of the evaluation follow.

4.3.1 English

Classifier	Precision			Recall		
	+	-	both	+	-	both
Baseline	89.25%	46.04%	69.38%	37.92%	16.65%	27.29%
Naïve Bayes	70.71%	72.47%	71.59%	70.71%	72.47%	71.59%
MaxEnt	75.87%	75.87%	75.87%	75.87%	75.87%	75.87%
SVM	89.60%	40.13%	64.87%	89.60%	40.13%	64.87%

Table 4.5: Evaluation results for English tweets.

Baseline results for English appear to be in line with what A. Go et al. had already established in their research (65.2%). Interestingly, there is a large disparity in accuracy between sentiment classes—positive messages were recognised over twice as often with similar increase in precision. This can probably be attributed to English having a broader vocabulary for describing positive emotions as established by A. Kramer [4].

Aforementioned disparity disappears in Bayesian classifiers, where performance between classes varies by less than 1pp. Our results hover appear to be on average 10pp lower than what A. Go et al. presented in their paper. This could be partially due to a larger data set (1,600,000 vs. 150,000 tweets), but also due to centring their queries used to create the test set around certain keywords, whereas I captured a snapshot of broader Twitter activity.

Maximum Entropy offers consistency similar to Naïve Bayes with accuracy and recall improved by 4pp. This makes Maximum Entropy the best choice for precise sentiment classification for English.

Support Vector Machines also show disparity in precision between positive and negative classes (89.60% vs. 40.13%) and subsequently average slightly lower than Naïve Bayes in both metrics. Our results for the positive class however outperformed previously quoted paper by 7.4pp.

4.3.2 Spanish

Classifier	Precision			Recall		
	+	-	both	+	-	both
Baseline	42.81%	77.44%	61.15%	22.78%	46.39%	34.59%
Naïve Bayes	66.37%	69.71%	68.04%	66.37%	69.71%	68.04%
MaxEnt	68.13%	71.47%	69.80%	68.13%	71.47%	69.80%
SVM	96.67%	25.33%	61.00%	96.67%	25.33%	61.00%

Table 4.6: Evaluation results for Spanish tweets

Baseline results for Spanish are similar to what we have established for English, with approximately 8pp difference in precision (in favour of English) and recall (in favour of Spanish). Remarkably, disparity between sentiment classes is reversed, with negativity being recognised more effectively. This can be due to cultural and linguistic differences between the two languages (e.g. different ratio of positive and negative words).

Using Naïve Bayes introduces a significant improvement in precision (6.89pp) and recall (33.45pp) and the gap in performance for different polarities have significantly shrunk.

Maximum Entropy performs only slightly better than Naïve Bayes with smaller disparity between results, making it the best performing classifier for this language.

SVM offers no improvement in precision over the baseline, but recall increases from 34.59% to 61.00%. On average, SVM performs worse than Naïve Bayes, but was surprisingly effective in classifying positive tweets.

4.3.3 German

Classifier	Precision			Recall		
	+	-	both	+	-	both
Baseline	39.45%	66.94%	53.63%	9.16%	16.57%	12.86%
Naïve Bayes	73.65%	62.65%	68.15%	73.65%	62.65%	68.15%
MaxEnt	75.07%	70.40%	72.73%	75.07%	70.40%	72.73%
SVM	95.87%	34.67%	65.27%	95.87%	34.67%	65.27%

Table 4.7: Evaluation results for German tweets

Baseline results for German are meagre compared to English or Spanish. With only 53.63% precision and 12.85% recall, it appears that bag-of-word dictionary count does not work very well in German. This is most likely due to the fact that on average 55.81% of the tweets will have been incorrectly classified as German by Twitter (as per Table 4.2).

Naïve Bayes brings classification performance to the level comparable to other languages, only 3.44pp short of what we seen for English. This means an improvement of over 600% in recall over the baseline.

SVM offers similar performance, but again seems to be biased towards positive messages (95.87% accuracy compared to only 34.67%).

4.3.4 Italian

Classifier	Precision			Recall		
	+	-	both	+	-	both
Baseline	71.35%	39.94%	54.58%	13.41%	8.60%	11.01%
Naïve Bayes	74.39%	63.79%	69.09%	74.39%	63.79%	69.09%
MaxEnt	77.20%	71.60%	74.40%	77.20%	71.60%	74.40%
SVM	96.00%	30.80%	63.40%	96.00%	30.80%	63.40%

Table 4.8: Evaluation results for Italian tweets

Baseline results for Italian are comparable to German, with the exception of reversed polarity bias.

Again, use of machine learning techniques offers a substantial improvement over the baseline, especially in terms of recall. Naïve Bayes performs fairly consistently for both polarities with approximately 10pp difference.

Maximum Entropy comes on top leading by over 5pp both in precision and recall.

Similarly to what we had seen before, SVM is substantially biased towards one sentiment class and subsequently performs slightly worse on average.

4.3.5 Polish

Classifier	Precision			Recall		
	+	-	both	+	-	both
Baseline	42.96%	69.05%	55.14%	6.23%	8.76%	7.50%
Naïve Bayes	70.49%	59.91%	65.20%	70.49%	59.91%	65.20%
MaxEnt	77.73%	66.53%	72.13%	77.73%	66.53%	72.13%
SVM	94.53%	36.00%	65.27%	94.53%	36.00%	65.27%

Table 4.9: Evaluation results for Polish tweets

Polish has the worst baseline accuracy of all evaluated languages with a disappointing 7.50% recall. This can probably be attributed to a high degree of inflexion which renders keyword-based approaches substantially less effective without stemming.

Naïve Bayes offers a nice improvement over the baseline, but still subpar compared to other languages.

Maximum Entropy offers a substantial improvement over Naïve Bayes and Support Vector Machines, leading by 7pp.

Support Vector Machine are highly polarised, but still manage to perform marginally better than Naïve Bayes (by 0.07pp) on average.

4.4 Discussion

All in all, machine learning techniques offer a substantial improvement over the baseline for all evaluated languages. Maximum Entropy appears to offer superior performance for all languages, with Naïve Bayes coming second, which seems to be consistent with the improvements seen by A. Go et al [1]. This introduces a tradeoff between computational performance and accuracy, as MaxEnt works substantially slower and requires more memory.

Performance seems to suffer proportionally to language classification accuracy as per Table 4.2. Baseline precision dropped by over 10pp as language accuracy

fell below 50%. Future research could therefore concentrate on how accurate language classification affects sentiment classification performance.

4.5 Summary

In this section I had shown that Twitter's native language filtering performs substantially worse for languages other than English and Spanish. I have also shown that whilst emoticons can be used as noisy labels for training data, the accuracy of this technique varies from language to language. I also established that machine learning techniques offer substantial improvements in classification accuracy for all evaluated languages with Maximum Entropy and Naïve Bayes offering the most consistent performance.

Chapter 5

Conclusion

5.1 Success

The purpose of this project was to evaluate different machine learning techniques for classifying Twitter messages in languages other than English. This goal has been achieved to an desirable level and provides a solid platform which could be commercially exploited. Our results for English were consistent with the literature and new results for other languages have been established. Moreover, I have shown that Twitter's built-in language filtering performs substantially worse for languages other than English, which inversely correlates to classification accuracy. I also demonstrate that consideration needs to be given to using emoticons as labels for sentiment classification due to variable accuracy of this approach.

5.2 Challenges

In retrospect, too much time was spent working on finalising the dataset without verifying the assumptions regarding the use of emoticons and built-in language filtering, which should have been an essential aim of the project. I therefore had to abandon the original aim to build an API for supplementing the Twitter stream with sentiment data and work on verifying the ground truth instead. Additionally, both Maximum Entropy and Support Vector Machines classifiers proven to be more difficult to implement than anticipated. Having said that, the experience gained in tackling those projects has proven invaluable to understanding the difficulty of successfully implementing machine learning techniques in real-life applications.

5.3 Scope for Further Research

This dissertation has only touched upon the possibilities of utilising the lingual diversity of Twitter in the field of sentiment classification and further research could be devoted into verifying alternative approaches and factors influencing accuracy of classification. An obvious extension to this dissertation would involve improving Twitter's native language classification. It could also explore alternative methods of collecting pre-labelled training data other than relying on emoticons.

Performance and scalability, and parallelisation in particular, is something that could be explored in more detail. As Twitter approaches handling thousands of tweets per second, further research could be carried out in distributed algorithms which have performance characteristics necessary to handle that amount of data.

References

- [1] Richa Bhayani Alec Go and Lei Huang. Twitter sentiment classification using distant supervision. 2009.
- [2] Ed H. Chi Aniket Kittur and Bongwon Suh. Crowdsourcing for usability: Using micro-task markets for rapid, remote, and low-cost user measurements. 2009.
- [3] Andrew Hunt. *Pragmatic Programmer*. Addison-Wesley, 1999.
- [4] Adam D.I. Kramer. An unobtrusive behavioral model of “gross national happiness”. In *CHI '10 Proceedings of the 28th international conference on Human factors in computing systems*, 2010.
- [5] Atebits LLC. “rt” vs “via”. round 2. <http://blog.atebits.com/2009/01/rt-vs-via-round-2/>.
- [6] Bo Pang and Lillian Lee. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of EMNLP 2002*, pages 79–86, 2002.
- [7] Jonathon Read. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *ACLstudent '05 Proceedings of the ACL Student Research Workshop*, 2005.

Appendix A

Project Proposal

Computer Science Tripos Part II Project Proposal

Sentiment Analysis for Multilingual Tweets

M. Drozdzyński, King's College

Originator: Dr Daniele Quercia

22 October 2010

Special Resources Required

File space on PWF – 2 GB
Developer Account at Twitter
Dataset of English tweets
Use of my own machine (MacBook Pro)

Project Supervisor: Dr Daniele Quercia

Director of Studies: Dr Simone Teufel

Project Overseers: Dr A. Copestake & Dr R. Harle

A.1 Introduction

Approaches for automatically classifying the sentiment (either positive or negative) of Twitter messages have been recently introduced. They are used by online shoppers to check the sentiment associated with products, or by companies to monitor the popularity of their brands. One limitation of those approaches is that they work for Twitter messages in English. This project considers the three most promising existing approaches (Naïve Bayes, Maximum Entropy, and SVM) and evaluates them for Twitter messages in a number of continental European languages. The three approaches will be made available through an API (similar to <http://twittersentiment.appspot.com/>).

A.2 Starting Point

This project builds upon two previously published papers on sentiment analysis of Twitter messages:

- Alec Go et al. – *Twitter Sentiment Classification using Distant Supervision:*
¹
- Hyung-il Ahn et al. – “How Incredibly Awesome!” – *Click Here to Read More*²

Significant research will need to be done around sentiment classification using Naïve Bayes, Maximum Entropy and Support Vector Machines. My prior experience in creating web applications and interacting with Twitter API will also be useful to this project.

The data set for English containing over 1,000,000 classified tweets has already been collected and will be provided by the Project Supervisor.

A.3 Substance and Structure of the Project

This project spans the domains of information retrieval, natural language processing and web development and can subsequently be divided into sections representing each of these three modules.

Work will commence by implementing a scraper which collect necessary training and test data from Twitter. The API provides an interface to query for Twitter messages in 19 different languages (Arabic, Danish, Dutch, English, Farsi

¹<http://www.stanford.edu/~alecmgo/papers/TwitterDistantSupervision09.pdf>

²<http://web.media.mit.edu/~hiahn/publications/ahn-et-al-icwsm2010.pdf>

/ Persian, Finnish, French, German, Hungarian, Icelandic, Italian, Japanese, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Thai) and allows for crawling tweets without violating any Terms of Service. The scope of this project will be limited to the following 5 languages, chosen to be spoken by either me or the Project Supervisor should a fallback to manual sentiment classification be required:

1. English
2. Spanish
3. German
4. Italian
5. Polish

The approach I describe will however be easily replicable to all remaining languages available via the API provided enough time to collect necessary training data.

In order to easily separate tweets into training sets of positive and negative sentiment, the scraper will run two individual queries for ‘:)’ and ‘:(’, with results from the former being qualified as positive and the latter as negative. Twitter API uses the following equivalence classes for emoticons: ‘:)’, ‘:-)’, ‘:)’, ‘:D’ and ‘=)’ are mapped to ‘:)’ and will subsequently be marked as positive; ‘:(’, ‘:-(’ and ‘: (’ are mapped to ‘:(’ and thus marked as negative. These emoticons will then be stripped off to avoid biasing the classifiers towards them.

As determined in previously cited paper on *Twitter Sentiment Classification using Distant Supervision*, this approach has been proven successful and will provide a working base for the classifiers. To further validate this approach, a random manual check will be carried out on 100 positive and 100 negative emoticons in each language to estimate accuracy of our training data set.

Further time will be spent studying the following classification algorithms: Naïve Bayes, Maximum Entropy and Support Vector Machines. I am hoping to be able to utilise existing solutions, hence research will be done into freely available implementations (including, but not limited to, *Weka*, *libsvm* and *SVM Light*) to verify if they meet the needs of this project.

Accuracy of classification algorithms will be measured by determining the number of correctly classified tweets. Two different metrics will be provided: first determined using leave-one-out and second 5-fold cross-validation, provided the latter could be determined.

Later in the project significant work will be done on creating a web application to utilise classification tools implemented in previous sections. The main goal of the application is to aid potential users in monitoring sentiment for particular Twitter queries and plot it against time or location. An API will also be provided which will enable running search queries against Twitter and returning results supplemented by sentiment analysis.

A.4 Success Criteria

Essential aims

For the successful completion of the project, the following criteria shall be met:

1. Collection of a limited number (1,000) of tweets in the following four languages: Spanish, German, Italian and Polish and implementation of feature reduction & duplicate filtering algorithms.
2. Evaluating the classifiers against the existing dataset of English tweets.
3. Preliminary results of classification using the limited dataset of Spanish, German, Italian and Polish tweets using the leave-one-out cross-validation technique.
4. API: Querying Twitter API for a given keyword or user and classifying returned tweets.
5. Application: Plotting sentiment for a given keyword or user over time.

Desirable aims

1. Collection of a larger dataset for Spanish, German, Italian & Polish
2. Classification of the extended dataset and evaluation using 5-fold cross-validation technique.
3. Application: Plotting sentiment for a given keyword or user over location.

A.5 Plan of Work

Aim

My aim is to consider and analyse three classification algorithms (Naïve Bayes, Maximum Entropy and Support Vector Machines) with respect to their accuracy in determining sentiments of multilingual tweets, as well as create an web application which can be used to plot sentiment of tweets over time or location.

Preparation

Research into relevant topics will be carried out in the first 4 weeks of the project. During this time a scraper will be written and preliminary data will be collected for all four languages.

Implementation

Approximately 10 weeks are allocated to the implementation of the project. An iterative methodology will be followed and upon successful completion of all iterations of the project all desired features will be included.

Evaluation

My project will be evaluated against the criteria set out in section 4 of this document. A successful project would meet the essential goals, and if possible the desirable ones.

Timeline and Milestones

The following timetable divides the time starting November 1st until the due date into 2-week iterations with individual goals to be achieved by the end of each block. Upon completion of these tasks, the final working project together with the dissertation will have been completed.

Michaelmas Term

Weeks 1 to 2: WB 01/11/10 and 08/11/10

- *Research:* Twitter API
- *Research:* Sentiment classification

- Set up development environment
- *Implementation:* Twitter scraper with a working duplicate filter. Successfully mine samples of 20 tweets per language

Weeks 3 to 4: WB 15/11/10 and 22/11/10

- *Implementation:* Feature reduction algorithms – parsing out Twitter specific properties (usernames, links), normalisation of repeated letters
- Collect preliminary data sets for all four languages
- *Research:* Naïve Bayes, Maximum Entropy, Support Vector Machines

Weeks 5 to 6: WB 29/11/10 and 6/12/10

- *Research:* Data mining and classification software
- Outline the overall structure of the dissertation
- Complete the Introduction and Preparation chapters of the dissertation

Christmas Vacation

Weeks 7 to 8: WB 13/12/10 and 17/01/11

- *Implementation:* Naïve Bayes
- Running the classifier against tweets
- Evaluation of accuracy metrics
- Section on Naïve Bayes in Implementation and Evaluation chapters

Lent Term

Weeks 9 to 10: WB 24/01/11 and 31/01/11

- *Implementation:* Maximum Entropy
- Running the classifier against tweets
- Evaluation of accuracy metrics
- Section on Maximum Entropy in Implementation and Evaluation chapters

- *Milestone:* 04/02/11 Progress Report due

Weeks 11 to 12: WB 07/02/11 and 14/02/11

- *Implementation:* Support Vector Machines
- Running the classifier against tweets
- Evaluation of accuracy metrics
- Section on SVM in Implementation and Evaluation chapters

Weeks 13 to 14: WB 21/02/10 and 28/02/11

- *Implementation:* API for querying Twitter API and supplementing the results with sentiment analysis
- Section on API in the Implementation and Evaluation chapters

Weeks 15 to 16: WB 07/03/11 and 14/03/11

- *Implementation:* Application for plotting sentiment of tweets over time and, if time permits, location.
- Section on the Application in the Implementation and Evaluation chapters

Easter Vacation

Weeks 17 to 18: WB 21/03/11 and 28/03/11

- Refinements and finalisation

Easter Term

Week 19: WB 25/04/11

- Evaluation chapter complete

Week 20: WB 02/05/11

- Conclusion chapter complete

Week 21: WB 9/05/11

- Formatting and finalisation of dissertation. Printed and bound by end of the week.

Week 22: WB 16/05/11

- *Milestone.* 20/05/11 Dissertation Due