Mikael Brevik & Øyvind Selmer

The best title in the world

Specialization project, fall 2012

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

Preface

Mikael Brevik & Øyvind Selmer
Trondheim, December 6, 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 Task Description

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. 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 Naive 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 Project goals

In this section we will describe our goals for this project. As this assignment was intended for a master thesis, we had to scope down the goals to fit the time schedule for a specialization project. We've scoped the assignment down to a set of goals that will make a good foundation for a master thesis.

1.2.1 Research on state-of-the-art sentiment analysis

A lot of work has already been done in the field of sentiment analysis, also when using the Twitter corpus. To be able to make a contribution to this work, we have to do research to gather knowledge about existing solutions and their performance. - Mention SLR here -

1.2.2 Server for sentiment classification

We will design and implement a highly modular python server with a basic form of classification. This system will work as a foundation for implementing the complete classification system in the proceeding project. For that reason we want it to be as modular as possible, to make it easy to swap out different parts of the system when necessary.

1.2.3 API layer with twitter API integration

Twitter offers a well documented REST API to obtain data from their corpus. To make our system easy to use for developers already using the Twitter platform, we will implement a web service that is compatible with the Twitter API. And as an extension to the already existing API we'll add a *sentiment* attribute to the returned tweet. This attribute should hold the result of the sentiment classification.

Chapter 2

Background Theory and Motivation

2.1 Background Theory

2.1.1 Sentiment analysis

Naive Bayes Classifier

The naive Bayes classifier(NBC) is a practical Bayesian learning model that are easy to understand and implement. For some classification tasks it has proven to be equally performing to more complex classifiers like artificial neural networks(ANN) and decision trees(DT)[ref]. NBC is used for learning cases 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 NBC is based on an assumption that all the attribute values are conditionally independent given the target value of the instance.

$$v_{NB} = P(v_j)\,\Pi\, p(a_i|v_j) \tag{2.1}$$

To classify an instance, the classifier's using the Maximum Likelihood Estimation(MLE) method to find the ratio of an attribute value and a given target value in the same instance in the training corpus. This means that it has to calculate the probability estimate P for each attribute a_i, given the target value. It then assigns the target value as the one that gives the highest product from multiplying all the probabilities P from the training data.

2.1.2 Twitter API

Twitter allows developers and others to access their data by the means of an application program interface (API). The API is implemented by using

Representational State Transfer (REST). REST is a style of software architecture for distributing data across the World Wide Web (WWW). By using the protocol HTTP's vocabulary of methods, developers can get, insert, delete and update data on Twitter, given the proper access. Some of the important methods are GET for retrieving data, POST for inserting, updating and sending data, DELETE for removing data. [Fielding, 2000]

Twitter uses OAuth for authentication. OAuth provides a way for clients to access resources on behalf of end users or other clients. OAuth is also used to provide third party applications access to a users data on a service, without sharing the users user name or password. [Hammer-Lahav, 2010]

After migrating to version 1.1 of their API, Twitter now has a limit on the number of requests an end user can do. As end users authenticate using OAuth, Twitter can identify them and limit their access if overused. The limitations are divided into 15 minute intervals. Some services on the API are limited to 15 requests per time window, other services are limited by 180 requests. E.g. the search service is limited by 180 requests, but the service for retrieving tweets from a Twitter list is limited by 15. [Twitter, 2012b]

Almost any aspect of Twitter is covered by the REST API. The most interesting parts are the ones to retrieve tweets in various ways. There are methods for retrieving entire user time-lines, mentions of a user, favourite tweets for a user, tweets based on geographical annotated locations, and so on. [Twitter, 2012a]

There are also real time streaming services in the API. Streams are not based on the Twitter REST API. For streaming a persistent HTTP connection is opened to the API. This way a client would not have to continuously poll the REST API to register changes or new data. A client would be provided by real time data from Twitter, but only have one single connection opened. [Twitter, 2012c]

The Twitter API uses JavaScript Object Notation (JSON) as their format for response. JSON is a lightweight data format often used as an alternative to XML. JSON was created by Crockford to be a subset of the JavaScript Programming Language but still be language-independent. JSON has a simple structure and aims to be minimal, portable and textual and has support for four primitive types (strings, numbers, booleans, and null) and two structure types (array and objects). [Crockford, 2006]

2.1.3 Node.js

Node.js is a platform built on Google's V8 JavaScript Engine. In later years Node.js as grown rapidly in popularity, this much due to it's scalability and event driven nature. As Node.js uses a asynchronous non-blocking I/O model, it is perfect for data-intensive real-time applications [Dahl, 2009]. Node.js is often refereed to as JavaScript on the server-side.

The original goal of Node.js was to make web sites be able to push capabilities. That is, for servers to be able to push states to the client, without the client having requested it first.

Since Node.js is JavaScript based and JSON is a subset of JavaScript, JSON is integrated seamlessly.

Although Node.js is a fairly new platform, it has a rapid growing user base. Even though Node.js was conceptualized not long ago, and it is not yet released as a stable version, there are about 19 000 modules released for it through the Node Package Manager (NPM) [NPM, 2012]. There are about 700 000 module downloads each day, using NPM.

2.1.4 Natural Language Toolkit for Python

2.2 Structured Literature Review

Chapter 3

Architecture

In this section we will describe the overall architecture and how the system works. First the general system will be described, and then the API Layer and Classification Server in turn.

To make the system as modularized and responsive as possible, the API layer was written in Node.js while the sentiment classification in the Python programming language. Both systems are continuously running servers. This allows us to have multiple services running, both for the API layer and the classifier, if needed for horizontal scalability.

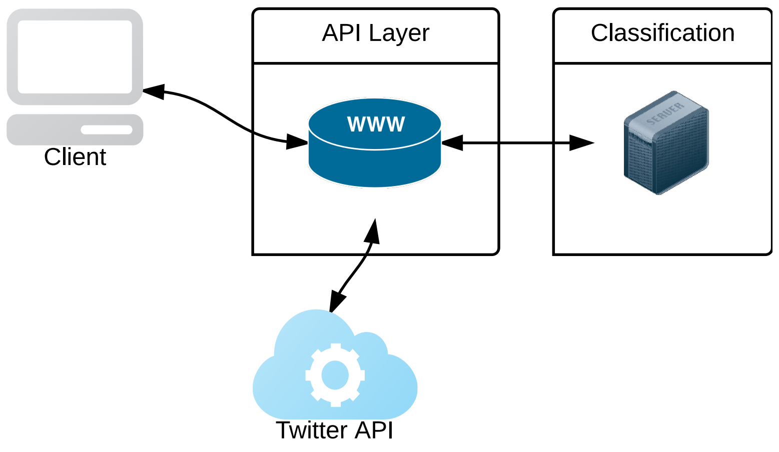

Figure 3.1: Architectural overview of the system. Client retrieves data from the Twitter API and uses the classification server to classify for sentiments.

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 HTTP requests, and 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 the client with the same JSON structure as the Twitter API sends out, only with an additional attribute noting the tweets sentiment. This architecture and application flow can be seen in Figure 3.1.

3.1 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 REST and Streaming API is easily manipulated and passed around.

The API layer works a thin layer extending the Twitter API. This means that the interface used by Twitter, with all defined options and appropriate methods, is reflected the same through the API Layer. This way all documentation for the Twitter API also documents our API Layer.

3.1.1 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 is 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 is processed by the classification server the classified sentiment is sent back to the TBH module and the original tweet object is extended to contain a property with the sentiment. When all tweets are classified, the TBH 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 3.2.

If there is an error during any part 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.

Every 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. In this event 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 TBH module opens a connection to the Twitter APi, but never close it. There is a continuously open connection to the Twitter server, which is feeding the TBH 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 is also never closed. This way there is an open connection between the client, and the API Layer and 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 having this entire process asynchronous, the system can process data independently of when it is published.

3.2 Sentiment Analysis Classifier

Python is computationally stronger than Node.js in many ways. Additionally, it is much more mature. There are a lot of stable, well documented, packages for handling various tasks. NLTK is one of these packages. As NLTK is a widely used package for natural language processing, and has integrated modules to handle lexicons, ngrams, and so on, it is a good choice for handling the sentiment analysis. As a dynamic typed language Python also allows for rapid development and prototyping. These attributes are some of the reasons Python is a good fit for this 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. A stringified tweet object is a serialized JSON object represented by a string. The classification server then converts this string to a Python dictionary. The response will be a string with the classification: either *positive*, *neutral* or *negative*. The classification scheme can be extended if necessary.

The classifier server can be initiated 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. If we wished to compare two different classification strategies, two servers could run side by side and a test framework could use the two servers and classify the same tweets and compare the results.

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 correlation with the API Layer.

3.2.1 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 using the strategy specified when initiating the server.

The strategy modules use various text filters, normalizations and feature extraction methods, depending on the selected strategy. The strategy modules can choose to have different filters and features for both the subjectivity classifier and the polarity classifier.

The strategy modules sends the features to the subjectivity classifier. This module classifies the tweet according to whether it is subjective or objective. Both classification types uses the classifier provider to retrieve a possible trained classifier. If no cached trained classifier exists, the classifier provider trains a classifier and caches the result for easier access next execution.

The subjectivity classifier will return response to the strategy module, which in turn will give return data to the API Layer, if the tweet gets interpreted as objective. This stops the classification cycle. However, if the tweet is classified as subjective, it is passed on to the polarity classifier module.

The polarity classifier works in a similar manner as with subjectivity. It uses the classifier provider to retrieve a classifier with the selected parameters. The polarity classifier provides the Strategy Module with the finished classification string.

3.2.2 Current Implementation

The current implementation uses a simpler form then the architecture described in this section. The classification is a simple, basic, approach, not using any machine learning. It uses the AFINN **?** lexicon to calculate sentiment strength. The sentiment is then return directly to the API Layer's Sentiment Provider, without using the subjectivity or polarity classifier.

The strategy modules has most of the responsibilities of the system. To explore different techniques for doing sentiment analysis, a new stra8tegy module can be developed using existing code for filtering and feature extraction.

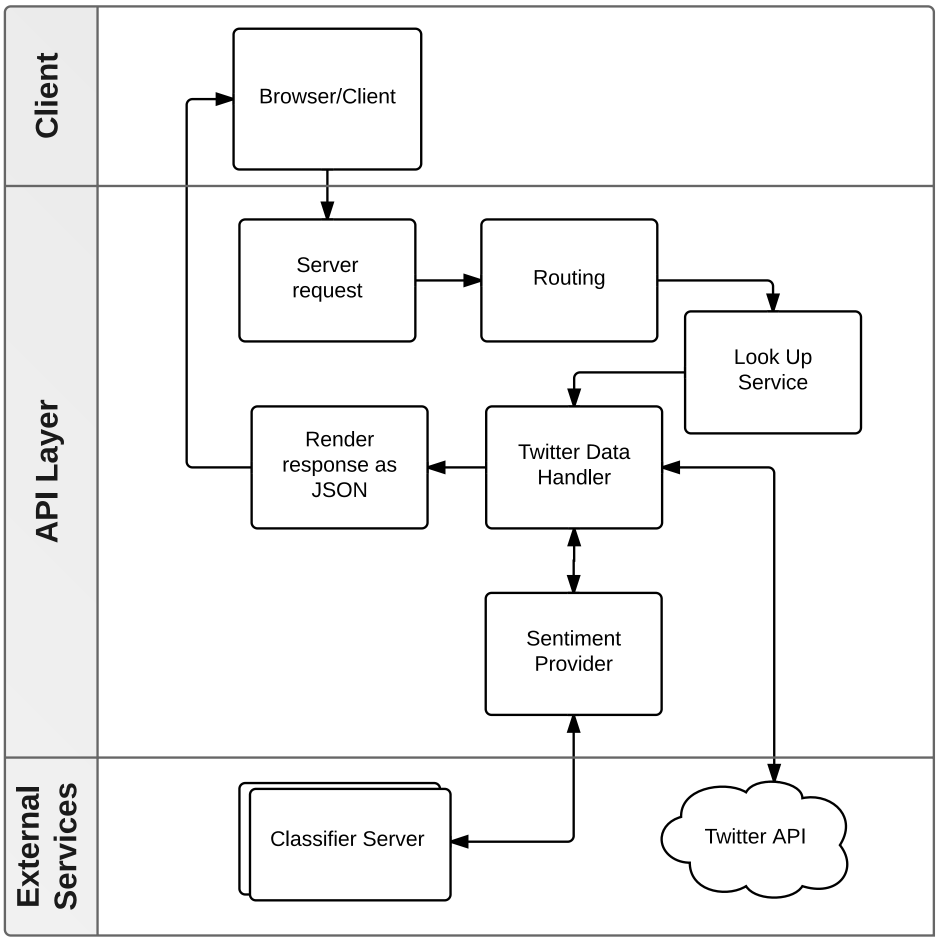

Figure 3.2: Architectural overview of the API Layer. A request is handled by the server, sending it to routing where it is processed and sent to service look-up. If a service is found a request is sent to the Twitter API and the received data is extended to contain a sentiment by the Twitter Data Handler module. When all of the Twitter data is extended, the data is given as a response to the requesting client.

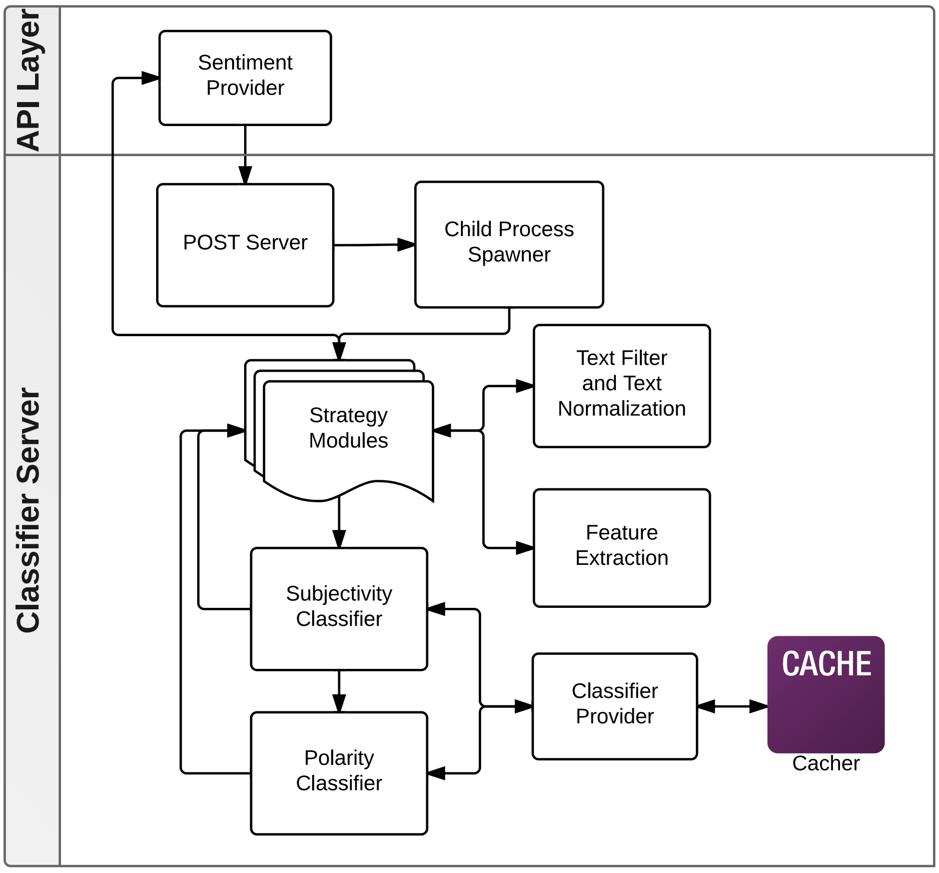

Figure 3.3: Architectural overview of the classification server. The API Layer makes a request to the POST Server. CHild processes is spawned per received tweet. Different Strategy Modules can be implemented and used. The strategies can use several text filters, normalization and feature extractions. The strategies use both subjectivity and polarity module to classify the tweet. After retrieving the sentiment it is provided to the API Layer

Chapter 4

Conclusion

4.1 Discussion

4.2 Contributions

4.3 Future Work

Bibliography

D. Crockford. The application/json media type for javascript object notation (json). 2006.

R. Dahl. Node. js: Evented i/o for v8 javascript, 2009. URL `http://nodejs.org/`.

R.T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

E. Hammer-Lahav. The oauth 1.0 protocol. 2010.

NPM, 2012. URL `https://npmjs.org/`.

Twitter. Rest api v1.1 resources, 2012a. URL `https://dev.twitter.com/docs/api/1.1`.

Twitter. Rest api rate limiting in v1.1, 2012b. URL `https://dev.twitter.com/docs/rate-limiting/1.1`.

Twitter. The streaming apis, 2012c. URL `https://dev.twitter.com/docs/streaming-apis`.

Appendices

4.4 Systematic Literature Review Protocol

4.4.1 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. For the fall project this protocol and the SLR in general, will be used for the authors to gain sufficient knowledge about sentiment analysis using the Twitter corpus.

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, is limited to a maximum length of 140 characters.

The goal of this fall 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 extending the API, the sentiment analysis data will be easy to use for existing Twitter developers, and already be heavily document 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. To uncover the different performances and how the problem has been solved by other researchers. This information will be used to implement a basic sentiment analysis application for the specialization project and to implement a more sophisticated application for the master thesis project.

4.4.2 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?

4.4.3 Search Strategy

The domain used for the search will be Google Scholar. Google Scholar aggregates results from different domains, and have 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 one group consists of words that are synonyms or have similar semantic meaning.

All search terms is placed in a table with the groups as columns and 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 4.1: Search terms and groupings

All results from 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.

4.5 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 is used to filter out the non-thematically relevant studies. The primary criteria is used on meta data such as title and abstract and 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.

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.

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 passed the primary and secondary selection criteria, will be passed on to the quality assessments.

4.5.1 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 is used in quality screening, to assess whether the papers includes a 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 is a clear statement of the aim of the research?

QC2 Is the study is 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 with?

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?

4.5.2 Data Extraction

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

- Name of author(s)
- Title
- Study identifier
- Year of publication
- Type of article
- Aims, objectives and contributions of study
- Name of system
- Type of machine learning algorithm
- User modelling technique
- Experimental design
- Test set source
- Training set
- 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.