View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by International Hellenic University: IHU Open Access Repository

INTERNATIONAL
HELLENIC
UNIVERSITY

Sentiment Analysis of Twitter Posts

Antoniou C. Georgios

SID: 3301120005

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

NOVEMBER 2013

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Sentiment Analysis of Twitter Posts

Antoniou C. Georgios

SID: 3301120005

Supervisor:	Prof. I. Vlahavas
Co-Supervisor	Dr. C. Berberidis
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

NOVEMBER 2013

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in ICT Systems at the International Hellenic University.

We are living in the information age, where a daily flood of messages of any kind takes place. The objective scope of this dissertation is to find a way to take advantage of these innumerable messages in order to discover what people like and what they dislike, what they prefer or how they feel about a particular product or topic.

To achieve this we developed a java program that can be used in order to build and use models suitable for giving sentiment ratings to sentences' parts. An attempt was made to combine more than one regression algorithms in order to create a more reliable and robust mechanism.

My thesis was completed under the supervision of Professor I. Vlahavas and Dr. C. Berberidis, to whom I owe heartfelt thanks for both their valuable assistance and guidance as well as for their overall contribution to my future career. On this occasion I would also like to thank Dr. E. Kontopoulos for his assistance, advices and support that he has given me throughout the implementation of this thesis. Finally, I would like to thank my family, my girlfriend and my friends who supported me and advised me in every step of my student life.

Antoniou C. Georgios

Thessaloniki, November 2013

Contents

Table of Figures

1 Introduction

Extracting information from Twitter is relatively easy. There is no need for the development and use of a special wrapper since Twitter offers a very powerful API for most of the currently used programming languages (e.g. Java, Python). This is very important, because it is very difficult to develop and especially maintain a wrapper to properly work after all possible minor changes that happen very frequently.

In the current dissertation an effort is being made to build a system that will be able to give a sentiment rating to pieces of sentences that have been extracted from their sentences based on their semantics. After experimenting a lot we decided to use a system that will integrate three different regression models using the Stacking method. A java application has been created for this reason that makes use of the publicly available algorithm implementations of Weka (Waikato Environment for Knowledge Analysis), which is a free suite of machine learning software written in Java, developed at the University of Waikato, New Zealand (Waikato, 2013).

This application is able to create different models, to constantly train them and of course to use each time the appropriate model in order to rate segments of sentences.

1.1 Capital Structure

In Chapter 2 it will be examined briefly what has been done so far in the field of Sentiment Analysis in Social Networks. An effort will also be made to discover the strengths and weaknesses in the literature, uncovering gaps which will justify the significance of the current work.

In Chapter 3 the requirements of the system will be elaborated and the application architecture will be briefly described.

In Chapter 4 the various components of the algorithm that was used will be justified and analyzed. Furthermore the results of the experiments that provide evidence in support of my thesis will also be presented in this chapter.

In chapter 5 an effort will be made to summarize what was learned and how it can be applied. The possibilities for future research and possible extensions will also be mentioned here.

2 Literature Review

2.1 Opinion mining / Sentiment Analysis

Whenever we want to make a decision, no matter how important it is, we seek the opinion of the others. Single people just ask for it, whereas organizations conduct polls, surveys, or hire special consultants. Terms “Opinion Mining” and “Sentiment Analysis” are used interchangeably; with the first one used more in the industry field. Sentiment analysis’ scope is to determine the attitude, view, emotion or judgment of a person concerning some specific topic or the overall contextual polarity of a document.

The biggest problem is that almost every document that we want to harvest is written in natural language text, which is unstructured data. Other problems rely on the fact that it is extremely difficult to discern between the nuanced usages of words (for example between “No way! The new iPhone 5S has a really convenient menu.” and “No way am I buying the new iPhone 5S!”). Extremely difficult is also to detect sarcasm or noisy data.

Most of the approaches try to classify a document after training appropriately some positive/negative classifiers and compare their results. In this case Sentiment Analysis is a classification problem, where the task is to classify documents into two categories (positive, negative). In (Pang, Lee, & Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning, 2002) the authors showed that three of the most used machine learning techniques (Naïve Bayes, Maximum Entropy Classifiers and Support Vector Machines) do not perform in sentiment analysis problems as well as in simple topic classification problems. In many approaches also a sentiment lexicon is used to determine which words/phrases have a positive or negative meaning. One of the most famous lexicons is the Harvard’s General Inquirer.

Something that matters a lot when trying to perform sentiment analysis is how subjective or objective a document is. Subjectivity is inextricably linked with emotions, views or beliefs. In another subsequent approach again from the same authors (Pang & Lee, A Sentimental Education: Sentiment Analysis Using Subjectivity, 2004) a machine learning method is proposed that makes use only of the subjective portions of a document that are manually pre-declared.

Something that is already happening, but is still in research phase, is trying to perform sentiment analysis on documents derived from all over the web and especially coming from Social Networks. If that could be successfully achieved, there would be no more need for conducting opinion polls.

In (Liu, Huang, An, & Yu, 2007) an effort is being made to assess the public's opinion from blogs in order to predict *product sales* performance. They focused on blogs containing reviews of products (e.g. movies). In their approach they focus mainly on the words that are sentiment related and try to make a step further from just classifying a movie as good or bad. Others (Park, Ko, Kim, Liu, & Song, 2011) or (Bakliwal, et al., 2013) have tried to perform opinion mining from news stories or tweets respectively in order to identify *political views*. In the latter approach an effort was also was made to take into consideration whether a tweet has sarcastic content or not (The annotators marked a tweet as sarcastic if its literal sentiment was different to its actual sentiment). Sentiment Analysis is also used in order to give predictive power concerning the various investors' choices (Dergiades, 2012).

2.1.1 Aspect-based Sentiment Analysis

According to (Liu B. , 2012) someone could examine a document in three possible ways. This would mean in case of Twitter and tweets that we could examine a tweet at the:

- *Tweet level*, is the specific tweet positive or negative in total?
- *Sentence level*, is each sentence positive or negative?
- *Entity and feature/aspect level*

Document level

The easiest of the three for mentioned approaches is surely the first one and it is what most current sentiment mining approaches at the moment do (simply classifying documents or posts like tweets as positive and negative). It does not take into account the different angles of the document, but it can have relatively good results especially in cases, where we are interested only in finding which documents does a specific user want to read (Antoniou, 2012).

Sentence level

There are many approaches that deal with the problem of sentence level and try to predict whether each opinion sentence is positive or negative (Zhang, Xu, & Li, 2012) (Hu & Liu, 2004). An opinion sentence is a sentence, which contains one or more opinion words. There are also some approaches (Brody & Elhadad, 2010) that although working at a sentence level, they employ a small number of topics, to automatically infer some aspects. This approach is a step away from working completely at an entity level.

Entity and feature/aspect level

Examining a document at an entity level is the approach that could lead to the best conclusions and performs more in-depth sentiment analysis. It is important to be able to understand the different aspects of a document (a tweet for example) reviewing a specific product or a document that compares two or three similar products. Imagine a tweet that speaks only about the new iPhone 5S. The twitterer may express a positive opinion about the phone battery, but in the same time a negative one about the phone's processor. The result of a regular sentiment classifier would be a classification as neutral or a score near zero (if the possible scores can also receive negative values) for the whole tweet. In this way, however an important piece of information gets lost.

Especially in the case of comparing different products the problem is even more serious. In this case a twitterer might say that iPhone 5S has the best battery ever, whereas HTC One S has the best display. Classifying this particular tweet simply as positive (because it contains two times the word "best") would lead to misleading conclusions.

Of course extracting these entities and aspects is a lot harder than simply determining sentiments and the combination of these two (extracting entities and determining sentiments) is even harder. One solution to the problem is the deployment of Ontology-based techniques (Kontopoulos, Berberidis, Dergiades, & Bassiliades, 2013). An ontology is an explicit and formal specification of a conceptualization (Gruber, 1993). It consists of terms (e.g. Smartphone), relationships between terms, properties, restrictions and various other statements. This way it provides a complete vocabulary to model successfully any domain. Ontologies constitute the principal component of Semantic Web.

The methodology that is followed in (Kontopoulos, Berberidis, Dergiades, & Bassiliades, 2013) is divided in two distinct phases. First of all, the domain ontology that will be used is built and then sentiment analysis is performed in tweets based on this particular ontology. The ontologies were built using the *Formal Concept Analysis* (Ganter & Wille, 1999), which applies a user-driven step-by-step methodology for creating domain models. The biggest advantage of this approach is that the ontology is built progressively according to the needs that are presented in the tweets. In the example of the tweet speaking of the properties of smartphones, this approach would not just find a ready ontology for smartphones with thousands of attributes and relationships, but it would create a new one based only on the given data set. This way we end up with a more data specific ontology with smaller size.

In another relevant paper (Nebhi, 2012) the author focuses just in the information extraction from tweets. He uses a ready ontology from DBpedia and creates an integrated disambiguation module based on popularity score and syntax-based similarity in order to select each time the right entity from the given ontology. That is an existing need, since simply extracting words out of a tweet is not always enough in order to classify it into the right ontology. There are many words (like "Washington" for example) that may refer to more than 90 different entities of the DBpedia ontology.

2.2 Sentiment Analysis in Social Networks

As we have already seen, with Sentiment Analysis we try to extract emotion or attitude out of a document or a piece of it concerning the whole document or a specific aspect of it. In social networks a document could be a tweet or a post at a social networking site like facebook or any piece of text possibly containing some opinion. If someone could, in some automatic way make sense of every possible piece of text in the web, he would be definitely the most powerful person on earth. People share opinions and personal experiences about any possible subject in the web. They submit reviews, write comments about various products or people (e.g. politicians, athletes etc.) or write articles in blogs, micro-blogs, forums and other forms of websites.

Performing opinion mining from micro-blogs is considered easier compared to other kinds of documents. The main reason for this is the short size of documents used in micro blogging, leading those most of the times to be straighter to the point. The most famous micro blogging service at the moment is Twitter and that's the reason that we are going to use it as a reference most of the times. Furthermore most of the research that has been done on Sentiment Analysis in micro-blogs up to now is around Twitter.

2.2.1 Existing Approaches

In (Barbosa & Feng, 2010) an approach is presented to automatically detect sentiments on tweets. The scope here is to classify the subjective tweets as a whole into three categories (positive, neutral, negative). It takes into consideration biased and noisy labels as input, since tweets due to the character restriction do not always contain syntactically consistent words.

Apart from syntactically inconsistent words and phrases that are used mostly in the oral speech, twitter users use special tags and smileys, which make it even more difficult to apply the same sentiment analysis approaches as those that are applied for other documents like reviews for example. In (Davidov, Tsur, & Rappoport, 2010) the authors utilize 50 twitter tags and 15 smileys as sentiment training labels. This way they take advantage of this peculiarity of Twitter messages and it is easier for them to give a highly positive rating to a tweet containing the smiley ";)" or the hashtag "#happy". But the paper itself makes it clear that it is far more difficult than it seems since people express often in the same sentence contrast sentiments and the hashtags do not seem to help a lot (e.g. "happy days of training going to end in a few days #sad #happy"). Another paper (Kouloumpis, Wilson, & Moore, 2011) attempts something similar by selecting and using only some hashtags that are used a lot and were deemed representative of the three categories (positive, neutral, negative) by the authors. As far address of the smileys problem mentioned before they choose to omit all tweets from the training process all tweets that contain both positive and negative emoticons.

Something that current research is deliberately ignoring until now is the reader's standing point. For example a tweet could be "I am so sad that Barcelona F.C. lost yesterday the game". The specific tweet gives a clear sentiment, but different readers could feel totally different about this.

2.3 Classifiers for Sentiment Analysis of Opinionated Text

2.3.1 Classifiers

There is a plethora of available machine learning-based classifiers that can be used for Sentiment Analysis. Some of the most popular data mining algorithms used for classification are according to (Wu, et al., 2007):

C4.5: Quinlan, J. R. 1993

It is used to generate a decision tree from a set of training data. It is an extension of another Quinlan's DM algorithm called ID3. In the Weka data mining tool C4.5 is implemented as J48

K nearest Neighbours (kNN): Hastie, T., Tibshirani, R. 1996

It is an algorithm for classifying objects based on closest training examples in the feature space. It is very easy both to understand and implement.

Naïve Bayes: Hand, D. J., Yu, K. 2001

It is a probabilistic classifier based on the well known Bayes' theorem. It is perhaps the easiest classifier to understand how exactly it works and it is used most of the times when someone wants to compare the effectiveness of a new classifier/method.

It can work fine on any dataset that can be shown as lists of features. A feature could be anything that can be present or absent. In the case of documents or tweets in general, features can be the words. It assumes that all variables are independent with each other, meaning that the presence or absence of a specific feature is totally unrelated to that of any other feature.

CART: Breiman, L., Friedman, J., Olshen, R., Stone, C. 1984

CART stands for "Classification and Regression Tree" and is used to refer to the two main types of decision trees used in data mining, Classification Trees and Regression Trees. This technique produces either a classification tree or a regression one depending on whether the dependent variable is categorical or numeric, respectively.

Support Vector Machines

SVMs are the combination of linear modeling and instance-based learning and are of course algorithms and not machines. A basic SVM takes a set of input data and predicts, for each given input, in which of two possible classes it belongs. Support Vectors are some critical boundary instances from each class. A great advantage of SVMs is that is more difficult to end up with overfitting situations, when using them. SVM algorithms for numeric prediction have also been deployed.

Regardless of the selection of the classifier there are two preponderant approaches that are used. The lexicon-based, that uses a lexicon of commonly used opinion words (e.g. good/bad), emoticons etc and the learning-based approach. The lexicon-based approaches can achieve high precision, but low recall. A significant factor for this low recall is the use of informal language in posts of micro-blogs like tweets. That's the reason that it has been a lot of effort in order to combine those two dominant approaches (Zhang, Ghosh, Dekhil, Hsu, & Liu, 2011).

2.3.2 Techniques to Improve Classification Accuracy

A lot of techniques exist to improve the efficiency of classifying algorithms like those described in the previous section. This is also known in statistics and machine learning as ensemble learning or ensemble methods. These techniques are used to combine various models learned from the data. In many cases it proves to be better not simply selecting the classifier with the most accurate results, but use more than one and combine their results. The three most dominant schemes at the moment, which can be used in both classification and regression problems are bagging, boosting and stacking, which are very briefly described below. The biggest disadvantage of all of them however is that it is very difficult to analyze them and indentify which exact factors led to the improvement in performance.

Bagging

In bagging we consider all classifiers used as equal. That means in the case of classification that the bagged classifier counts the votes of the independent classifiers and assigns the class with the most votes. In the case of regression or prediction it gives as an output the average value of all predictions. Something that holds is that the more votes from independent classifiers, the more reliable the results are. It turns out that the combined model from bagging can be better or not than the original model, but it is never worse. One restriction of bagging is that it must combine models of the same type (e.g. decision trees).

Boosting

It is perhaps the most powerful technique among the three most prominent. It has a lot of similarities with bagging, with the difference that weights are assigned to the various models that form the boosted classifier. It pays attention especially to the training tuples that are misclassified. This way it can achieve higher accuracy with the risk nevertheless to over-fit a specific dataset.

Stacking

Stacking is less widely used than the other two techniques. The main reason is that it is even more difficult to be analyzed and there are no commonly accepted proper ways of using it. The biggest advantage of this particular technique is that unlike the other two approaches, it can combine models that are not of the same type (e.g. only decision trees). Stacking introduces the notion of the metalearner, which is another algorithm used to decide which is the most reliable of the classifiers used and how to combine in the best way the outcomes of the other models.

2.4 Existing Systems – Similar Applications

In this section applications that perform in some way sentiment analysis for social networks will be examined. Two are the biggest weaknesses of all of them that our approach in the current dissertation will try to solve.

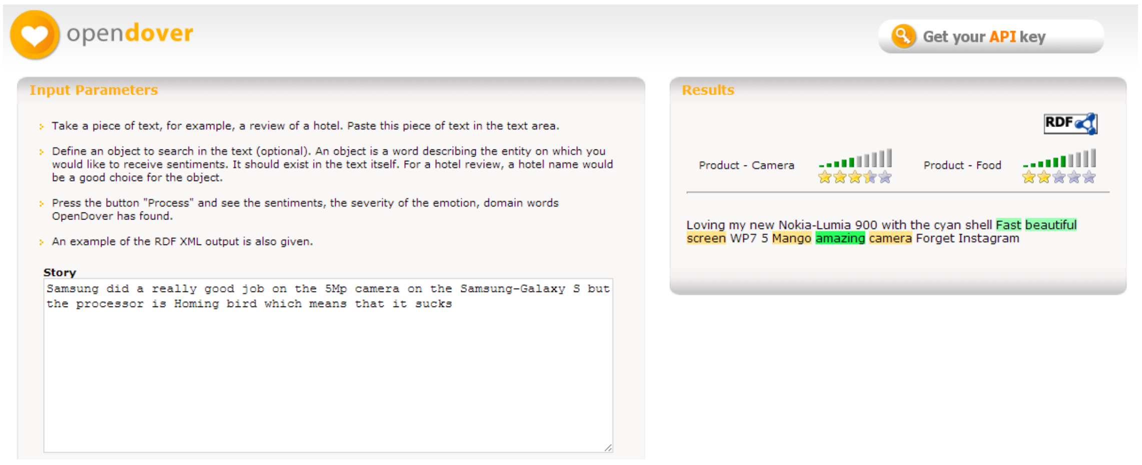
The first weakness is that all of these web apps do not reveal exactly how they work or which exact methods they use in their opinion mining process. For example some of them say that they use a sentiment lexicon, without coming into more details like which exact lexicon or how many words does it contain. Such methods could also be the classifiers that were used and their exact set-up. That means that they operate for the end user as a black box that he has to trust, without even know some basic metrics for the evaluation of the application, like the percentage of correctly classified instances or the percentage of a relative absolute error.

The second weakness is that most of the existing web applications examine and classify the document (this could be a tweet or a post) as a whole. They do not take into account the different aspects that it may contain. A different approach related to this particular weakness is attempted by OpenDover, which gives a different rating to some specific predefined aspects. For example it has a pre-built ontology for hotel reviews, so in the sentiment analysis of a hotel review it can give a different rating for the hotel, the restaurant and the flight.

2.4.1 Open Dover

Open Dover is a web service that performs sentiment tagging on any piece of text. The user provides the text that it wants to be analyzed. The content is returned as feedback with sentiment tags and negative/positive appraisal of the whole document. In order to achieve this, OpenDover makes use of a knowledge base of opinion words, domain-related words and intensifiers and distinguishes between context dependent and context independent opinion words. Furthermore OpenDover tries to indentify some commonly used domains and evaluates a piece text taking into consideration these domains. Unfortunately the result is not always the desired one, since there is a limit in the available domains ready to be used.

In the following picture we can see an example of how OpenDover works and what it returns.



- Take a piece of text, for example, a review of a hotel. Paste this piece of text in the text area.
- Define an object to search in the text (optional). An object is a word describing the entity on which you would like to receive sentiments. It should exist in the text itself. For a hotel review, a hotel name would be a good choice for the object.
- Press the button "Process" and see the sentiments, the severity of the emotion, domain words OpenDover has found.
- An example of the RDF XML output is also given.

```
Samsung did a really good job on the 5Mp camera on the Samsung-Galaxy S but
the processor is Homing bird which means that it sucks
```

Product - Camera ★★★★☆ Product - Food ★★☆☆☆

Loving my new Nokia-Lumia 900 with the cyan shell Fast beautiful screen WP7 5 Mango amazing camera Forget Instagram

Figure 1 - OpenDover Application

More about OpenDover you can find in (B.V., 2013)

2.4.2 Tweetviz

TweetViz is a web-tool that retrieves tweets containing some specific keywords provided by the user. It places them along two axes, one describing pleasant versus unpleasant, and the other describing how strong the sentiment is. It offers eight different ways of presenting the for-mentioned tweets, including presenting them according to their sentiment or their timeline, or as a tag cloud.

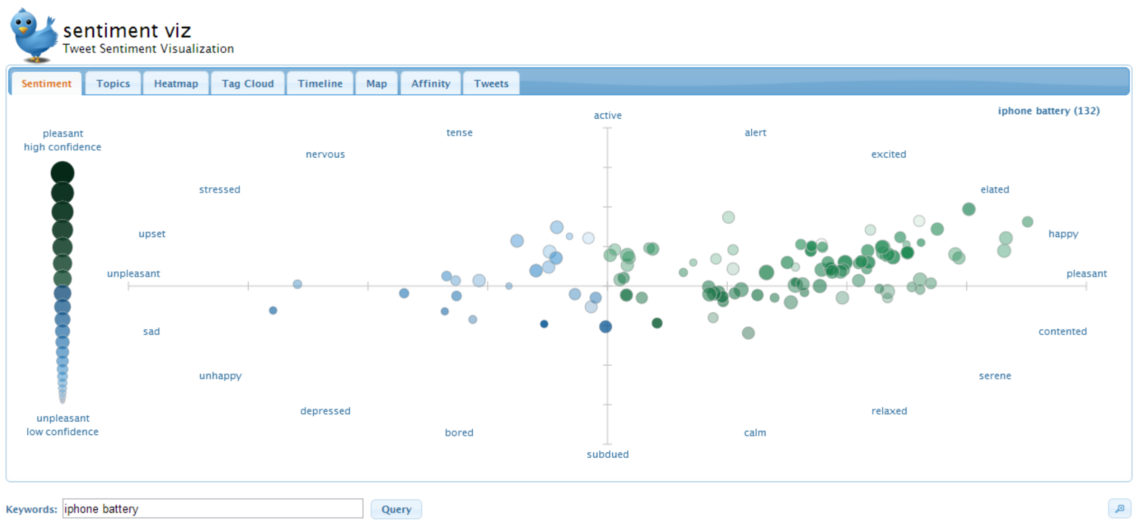


Figure 2 - TweetViz Application

An estimate of the total emotion contained in a tweet is calculated and each tweet is shown as a circle positioned by sentiment. As we see in picture 2 unpleasant tweets are colored blue on the left whereas pleasant ones are colored green on the right. Sedate tweets are darker and active tweets are brighter on the top.

In order to estimate the sentiment of the tweet, a sentiment dictionary is used. Each tweet is being searched for the specific words that are contained in the dictionary. The words' pleasure ratings are then combined to give the overall rating of the whole tweet.

It doesn't take into account sarcasm and that's the reason that tweets like "Love how my iPhone battery skips from 20% down to 3% for sending one text" are classified as "happy". Furthermore it does not take into account the different aspects of a tweet. For example the tweet "I love my iPhone, but the battery life sucks." is simply classified as "elated" despite the fact that we were searching for tweets having to do with the iPhone battery (meaning that we were mainly interested in the second half of the above tweet, which undoubtedly has a negative meaning).

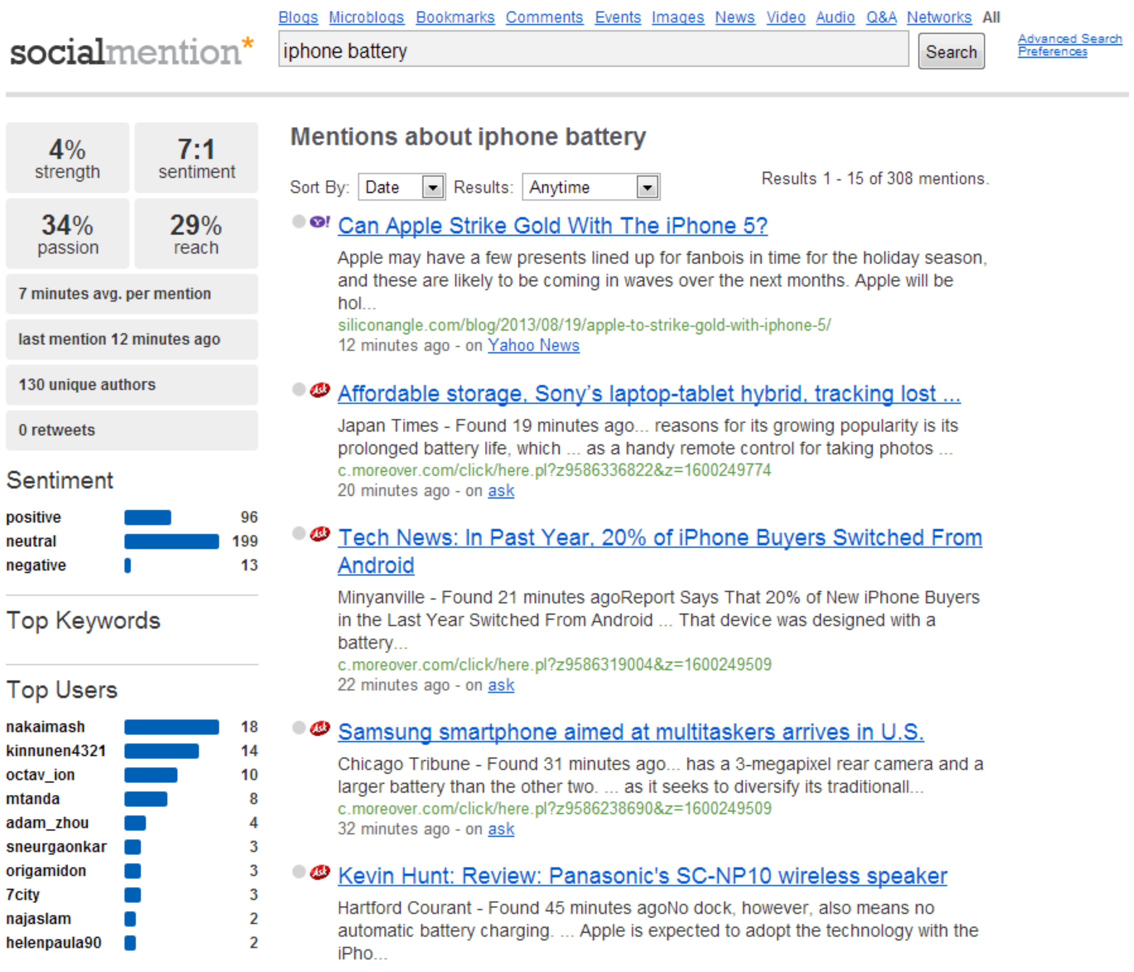
More about TweetViz you can find in (Tweet Sentiment Visualization).

2.4.3 Sentiment140

SEE PAPER Sentiment Knowledge Discovery in Twitter

2.4.4 Social Mention

Social Mention is a web platform that aggregates content from many different sources like facebook, Twitter, Ask, Google, YouTube etc. It tries to measure what people think about a particular topic in the web. As we can see in picture 3 an effort is also made to classify all the reports that are retrieved into 3 sentiment categories (positive, neutral, negative). Regardless of the topic of the query it seems that almost always more than the half results are classified as neutral. There is no information available about how exactly this sentiment is calculated. Finally it provides an API for developers, who want to integrate these functionalities in some web application.



socialmention*
Blogs Microblogs Bookmarks Comments Events Images News Video Audio Q&A Networks All
iphone battery
Search
Advanced Search
Preferences

4% strength
7:1 sentiment
34% passion
29% reach
7 minutes avg. per mention
last mention 12 minutes ago
130 unique authors
0 retweets

Sentiment

positive	96
neutral	199
negative	13

Top Keywords

Top Users

nakaimash	18
kinnunen4321	14
octav_ion	10
mtanda	8
adam_zhou	4
sneurgaonkar	3
origamidon	3
7city	3
najaslam	2
helenpaula90	2

Mentions about iphone battery

Sort By: Date Results: Anytime Results 1 - 15 of 308 mentions.

Can Apple Strike Gold With The iPhone 5?
Apple may have a few presents lined up for fanbois in time for the holiday season, and these are likely to be coming in waves over the next months. Apple will be hol...
siliconangle.com/blog/2013/08/19/apple-to-strike-gold-with-iphone-5/
12 minutes ago - on Yahoo News

Affordable storage, Sony's laptop-tablet hybrid, tracking lost ...
Japan Times - Found 19 minutes ago... reasons for its growing popularity is its prolonged battery life, which ... as a handy remote control for taking photos ...
c.moreover.com/click/here.pl?z9586336822&z=1600249774
20 minutes ago - on ask

Tech News: In Past Year, 20% of iPhone Buyers Switched From Android
Minyanville - Found 21 minutes agoReport Says That 20% of New iPhone Buyers in the Last Year Switched From Android ... That device was designed with a battery...
c.moreover.com/click/here.pl?z9586319004&z=1600249509
22 minutes ago - on ask

Samsung smartphone aimed at multitaskers arrives in U.S.
Chicago Tribune - Found 31 minutes ago... has a 3-megapixel rear camera and a larger battery than the other two. ... as it seeks to diversify its traditionall...
c.moreover.com/click/here.pl?z9586238690&z=1600249509
32 minutes ago - on ask

Kevin Hunt: Review: Panasonic's SC-NP10 wireless speaker
Hartford Courant - Found 45 minutes agoNo dock, however, also means no automatic battery charging. ... Apple is expected to adopt the technology with the iPho...

Figure 3 - Social Mention Application

More about SocialMention you can find in (Cianciullo).

3 Problem Definition and Architecture

There is no single machine learning model that is appropriate to every data mining problem. The same holds also for sub problems like mining Twitter. Data mining is an experimental process. That's the reason that we compared lots of different schemes to find the most suitable for classifying Tweets into specific ontologies and after that give a rating to each different aspect that is contained into them.

Our objective scope is to provide a working mechanism, which will take as an argument a phrase or a sub-sentence and it will return a sentiment score to this particular part of speech. A prerequisite for this to happen is the development of another mechanism that will take a piece of text (a tweet for example), and will break it into as many sub-sentences as the different ontologies that are contained in it. This is something that is already the subject of research of another dissertation that is conducted at our university alongside this current dissertation.

3.1 Ontology-based Opinion Mining / Sentiment Analysis

It is very important to distinguish the different aspects of a piece of document and give a distinct rating for each different aspect of it. A tweet's content could be for example: "Disappointed about the minimum focal distance of my Nokia-Lumia 800s camera, but its screen display is simply superb". If we don't follow an ontology-based opinion mining approach, and regardless of the technique that will be used, we will probably end up with a neutral rating of this particular tweet, since in the first part of the tweet a clearly negative sentiment is expressed in stark contrast of the positive sentiment of the second half. That is certainly not the desirable result.

Using ontologies would have as a result firstly to find out that the whole tweet refers to an ontology named for example “Mobile Phone” or “Cell Phone” or something similar. For this reason we could make use of the equivalent classes in OWL. Subsequently we should indentify classes that are connected somehow with the main superclass named for example “Cell Phone”. These classes could be subclasses of the superclass or can be related to the main class via various properties. In the first case two possible subclasses of the “Cell Phone” class could be “Nokia Phone” and “HTC phone” or “Windows Phone” and “Android Phone” depending on the respective ontology used. In the second case (where classes are related via properties), a “Cell Phone” may consist of a battery, a screen, a camera and many other components. “Battery”, “Screen” and “Camera” are classes that are related to the “Cell Phone” class via the “Consists of” property.

After indentifying to which ontology or ontologies the tweet that has to be evaluated belongs, we would like to give a distinct rating to every class that is included. In the case of the preceding example we would like to end up with two different ratings for the two components and perhaps a third rating for the specific Nokia Phone as a whole. To do so we must find a way as already mentioned to break the initial tweet in semantic portions and use each of them as an argument to a classifier that will be used to return this way a semantic rating.

It will not be examined in this work how the ontologies that will be used are created. An effort will only be made to implement a custom methodology that will extract sentiment from the distinct features/aspects of a tweet.

3.2 User - System Requirements

An attempt is made to create an application that will be able to give a sentiment rating to various pieces of text that will be given to the application as an argument. There are two main approaches to achieve this.

1. The first one relies on counting sentiment-related words and optionally give weights to them. These words come from various sentiment lexicons (dictionaries of words) that have been created and evaluated by experts. Such widely used lexicons are for example Harvard Inquirer (Stone, Dunphry, Smith, & Ogilvie, 1966), SentiWordNet (Esuli & Sebastiani, 2006), WordNet (Miller, 1995) and

Micro-WNOp (Cerini, Compagnoni, Demontis, Formentelli, & Gandini, 2007). In the case of the Twitter these lexicons can be enhanced with a list of emoticons that express positive or negative sentiments.

2. The second one relies on learning the classifier that will be used on previously rated text pieces. This way a more targeted approach can be achieved. This is the approach that is adopted in most recent papers and books like for example in (Kumar, Morstatter, & Liu, 2013).

Although the first approach is the simplest one, we will use the second approach for building our application. This way we can overcome the big disadvantage of simply using a sentiment vocabulary. This disadvantage is that various sentiment lexicons are built for different purposes and for this reason they contain specific words. But even in the case where they use the same words, these words can have totally different meaning. For example the word "bomb" usually has a negative meaning but in the case of mobile phone reviews it has a positive one ("The new i-phone 5S was the bomb!"). So this way we would need a different sentiment vocabulary for almost every different model that we would build. This is of course something very difficult for someone to achieve.

But even using the second approach we should built different models for different ontologies that we want to semantically evaluate.

3.2.1 Typical Usage Scenario

In this section it will be described how the current application is intended to work in order to achieve the best possible results.

As mentioned before, a different model should be used for totally different semantic entities. So, we should firstly classify every entity that is found to a group according to the semantic domain that this particular entity belongs to. For example if the entity Nokia-Lumia Camera occurs, the tweet that contains it should be classified to a group named "Technology". Other groups can be "Health", "Sports" and "Music". For this purpose we could make use of the ready categorization that Twitter uses when someone searches who to follow according to desired topics. For each of these groups a separate model will be trained and used subsequently.

After having collected a significant number of tweets, divided into pieces of text containing different aspects, the training of the classifier begins. All pieces of text that are grouped together are used as a training set to build a model. In this phase the user of the application should also provide a rating to each of these portions of text. This is of course a time consuming procedure but it is the safest method to train every model exactly as someone wants.

After the initial training the user will be able to save the created model.

Furthermore the user will be able to load any previously created model and re-evaluate it on another set or another instance in order to:

- give new sentiment scores to portions of text that refer to specific entities.
- further train the existing model if that is needed.

3.2.2 Functional Requirements

FR-1: The output sentiment score has to be given on a scale from one (1) to six (6), where one refers to the most negative possible sentiment score and six refers to the most positive possible sentiment score.

- User priority: 5
- Technical Priority: 3

Justification: A simple characterization as positive or negative is not enough. We need a larger gradation. It is important to distinguish between the totally negative and a less negative rating. And of course it is important to have also a rating (in our case this is 3.5) for neutral sentiment rating.

FR-2: The user should be able to further train any model at any time.

- User priority: 5
- Technical Priority: 5

Justification: The user needs to know that he has the ability to improve the performance of the classifier, when deemed appropriate.

FR-3: The system should be operated for a long time without problems.

- User priority: 4
- Stability: 5

Justification: The volume of data is such that their processing takes time.

FR-4: The model that will be used for training and rating should work reliably in spite of textual errors.

- User priority: 5
- Stability: 4

Justification: This is very important since it is very often for documents in social media to contain misspellings, abbreviations, slang language etc.

FR-5: The system must be efficient and it must consume as little storage and processing power and time as possible.

- User priority: 5
- Stability: 4

Justification: This is needed because of the very big volume of documents that are going to be handled.

FR-6: The user should give as arguments to the algorithm a phrase and optionally a rating. The algorithm should decide based on that input if the user wants to train a model or if he requests from the algorithm a rating.

- User priority: 5
- Stability: 5

Justification: The user should be facilitated. He should be able to use the same mechanism both for training and rating.

3.2.3 Non-Functional Requirements

NFR-1: The system must be able to respond in a reasonable time depending on the stimuli it receives.

- User priority: 5
- Technical Priority: 5
- Stability: 5

Justification: The system must be fast.

NFR-2: It must be easy to use.

- User priority: 5
- Technical Priority: 3
- Stability: 4

NFR-3: Must be accompanied by a short manual that will describe and document all individual system functions.

- User priority: 5
- Technical Priority: 2
- Stability: 5

Justification: The aim is to facilitate those users who are not so familiar with such applications.

3.3 Architecture

The architecture of the application that is implemented in the current dissertation is presented in the following diagram.

The following diagram shows briefly the steps used to generate and evaluate the various models that are needed. At first the training data set is loaded in the form of an .arff ASCII text file. In such files a list of instances sharing a set of attributes is described. ARFF files are used with the Weka machine learning software (Waikato, 2013).

Afterwards we have to apply a filter to our data that will break the initial text into individual words or N-Grams. This is absolutely necessary since most classifiers are not capable of handling directly string attributes. To do so we use the FilteredClassifier class based on Weka 3.6. This class applies a classifier to the data after previously having applied an unsupervised filter. In our case this filter is StringToWordVector. All the architecture's components, as well as why they were selected, will be analyzed in the next chapter (Contribution and Implementation).

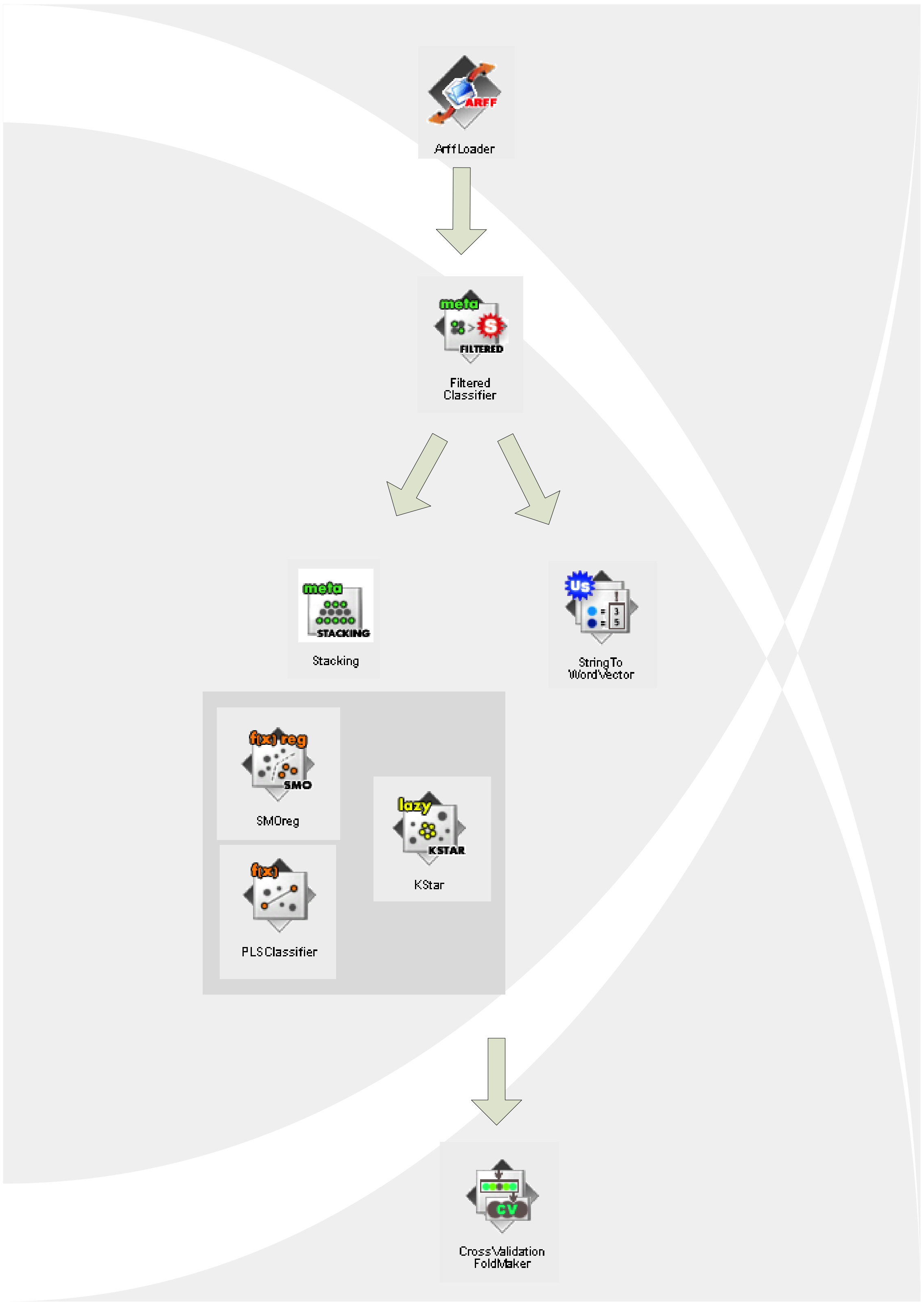


Figure 4 - Sentiment Analysis Application's Architecture

4 Contribution - Implementation

In the current chapter there will be an attempt to analyze how exactly the application, which is used to give sentiment ratings to tweets, works. Additionally every component choice will be completely justified. Finally the results of the experiments that provide evidence in support of my thesis will also be presented in this chapter.

4.1 Design and implementation

4.1.1 Regression vs. Classification

The first dilemma that we had to face was to choose if our problem was better described as a regression or a classification problem. In regression problems we predict a response and the output variable takes continuous values. In classification problems the output variable takes class labels. In our case we want to give a rating to a piece of text that refers to a specific entity. If we address the problem as a regression one, we will have as a result any real number from one to six (as defined in the first functional requirement).

By contrast if we address the problem as a classification one, the input argument (in our case the part of the tweet that refers to a specific entity) will be classified in one of the following classes: “1”, “2”, “3”, “4”, “5” or “6”. What we should and took care of is the fact that rating is not simply a categorical variable but an ordinal one. When we refer to an ordinal variable, we mean a variable where the order between values matters. The difference between the values on the contrary is not that important (This does not hold in our case where we want to give a rating, so the difference between the values plays a significant role). If we asked for instance mobile phone users to evaluate their experience in a scale from one to six, a rating of 5 is better than a rating of 4. This means that well known classifying algorithms such as Naïve Bayes, kNN, C4.5 and many others cannot be used in this case.

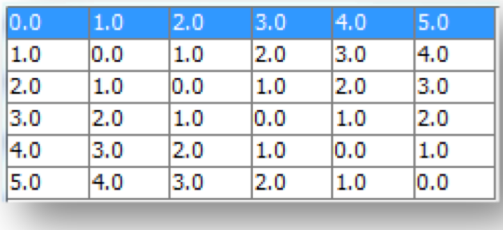
There are however two proposed techniques to handle ordinal variables. These are:

1. A Meta classifier called CostSensitiveClassifier.
2. A Meta classifier called OrdinalClassClassifier (Hall & Frank, 2001).

We have tested both of the above classification techniques but unfortunately the results were not the expected ones.

In order to test the effectiveness of the above two classification techniques we had first to convert the rating attribute from a numeric one to a nominal one. We did this using the NumericToNominal Filter of Weka library. This way we created the seperated_tweets_OneToSix – Nominal.arff file that was used for the evaluation of the classification algorithms.

In order to use properly the Cost Sensitive Classifier we had to define the following cost matrix.



0.0	1.0	2.0	3.0	4.0	5.0
1.0	0.0	1.0	2.0	3.0	4.0
2.0	1.0	0.0	1.0	2.0	3.0
3.0	2.0	1.0	0.0	1.0	2.0
4.0	3.0	2.0	1.0	0.0	1.0
5.0	4.0	3.0	2.0	1.0	0.0

Figure 5 - Cost Matrix used in the Cost Sensitive Classifier

This way we declared that we are not interested only in the percentage of correctly classified instances but we are interested even more to see to which class a misclassified instance is classified. For both cases we wouldn't bother at all if for example only 30% of the instances were correctly classified and all the rest were classified in a neighbor class. For example if a training instance had a rating of 'four' we would expect that with the use of the above algorithms it would get a rating of three or five (if not ideally four).

Using the CostSensitiveClassifier on a J48 tree we get 48.6% correctly classified instances. This is a high percentage taking into account that we have six distinct classes. Similarly using the OrdinalClassClassifier on a J48 tree we get 45.2% correctly classified instances, which is also a very high accuracy percentage.

The reason why we did not prefer one of these two solutions for our solution is displayed in the next two figures. In these figures we observe that there is a dispersion of the misclassified instances in all the remaining five classes. We see for example in the confusion matrix of the Cost Sensitive Classifier that the instances that should be rated with “one” don’t get classified only to neighbor classes (“two” and “three”) as we would expect. On the contrary we observe many instances that are classified in classes “five” (9 instances) and “six” (11 instances).

Something similar holds when using the Ordinal Class Classifier as we can see in figure 7.

a	*b*	*c*	*d*	*e*	*f*	<-- classified as
34	10	0	1	9	11	| a = 1
8	**15**	3	2	11	22	| b = 2
5	9	**0**	0	8	8	| c = 3
12	2	1	**22**	40	23	| d = 4
9	4	0	12	**102**	85	| e = 5
3	5	2	6	52	**170**	| f = 6

Figure 6 - Confusion Matrix of the CostSensitiveClassifier

a	*b*	*c*	*d*	*e*	*f*	<-- classified as
15	21	4	9	15	1	| a = 1
4	**16**	8	10	18	5	| b = 2
2	1	**2**	9	14	2	| c = 3
1	8	1	**32**	43	15	| d = 4
0	6	10	35	**109**	52	| e = 5
0	5	6	22	60	**145**	| f = 6

Figure 7 - Confusion Matrix of the OrdinalClassClassifier

4.1.2 StringToWordVector Filter

There is no way to use directly a string and take advantage of it in order to come up with useful results. We need to analyze it and break it into individual words or N-Grams (we chose this alternative). The regression algorithms will use these attributes together of course with the rating attribute (if we speak of the training phase) in order to create and evaluate models. So whereas the user feeds the algorithm with only two attributes, the phrase that needs to be rated and optionally a rating, the algorithm as a matter of fact uses hundreds of attributes.

We used WEKA's text filter “StringToWordVector” to convert string attributes into a set of attributes representing word occurrence information from the text contained in the strings. We are interested only in which terms occur and not how many times each one occurs. Pang et al. showed that simply term presence achieves better results than word frequency in problems having to do with sentiment analysis (Pang, Lee, & Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning, 2002).

Furthermore we turn all N-Grams into lowercase. This way we avoid mix-ups. Almost all of the current studies use this technique although someone could claim that a word in capitals express a different sentiment from the same word written in lowercase. This is true but it is very difficult to benefit from this fact.

NGram Usage vs Stanford’s CoreNLP

Simply observing which words are contained in a sentence is not enough. One easy way to prove that is the use of the adverb “not”, which can change completely the sentiment of a whole phrase. For example let’s take the phrase “My dog is not the best dog in the world”. This phrase would get a sentiment rating around four (in a scale from one to six). On the contrary the same phrase without the adverb “not” would get a rating of six. In a reverse case where the main adjective of the phrase had a negative meaning the use of the adverb “not” increases the sentiment rating of the phrase. The phrase: “My dog is not the worse dog in the world.” should take a sentiment rating around 3 whereas the same phrase without the adverb “not” would take the absolute negative rating. This means that just the presence of this specific adverb does not necessarily imply a negative sentiment.

Our first approach to tackle this issue was to use a Natural Language Processing tool like Stanford’s CoreNLP (Stanford University, 2013). Unfortunately this was proved to be more complicated than it firstly seemed. We used the CoreNLP’s ParserAnnotator. We wanted to examine some simple grammatical relations like the negation modifier that we mentioned before. This modifier is the relation between a negation word and the word it modifies.

After examining the phrase of our previous example (“My dog is not the best dog in the world.”) we ended up with the following typed dependencies:

```
poss(dog-2, My-1)
```

```
nsubj(dog-7, dog-2)
cop(dog-7, is-3)
neg(dog-7, not-4) (which is undesirable ...)
det(dog-7, the-5)
amod(dog-7, best-6)
root(ROOT-0, dog-7)
prep(dog-7, in-8)
det(world-10, the-9)
pobj(in-8, world-10)
```

The results were not the expected concerning the sentiment analysis of a phrase. We would like to come up with a dependency between the adverb "not" and the adjective "best". We faced analogous problems with other kinds of dependencies too.

That's why we decided to make use of N-Grams. Using N-Grams is relatively easy and offers to our algorithm an ability to scale up. Furthermore this way we end up with a small, robust, fast and relatively credible system.

In our system, we use unigrams (ngram of size 1), bigrams (ngrams of size 2), and tri-grams (ngrams of size 3). N-grams of length one to length three simultaneously. Below are presented a few of the N-Grams that our model built during the training process that have to do with the adverb 'not'.

```
@attribute 'not a better' numeric
@attribute 'not a fan' numeric
@attribute 'not a lot' numeric
@attribute 'not all' numeric
@attribute 'not all hyped' numeric
@attribute 'not all that' numeric
@attribute 'not bad' numeric
@attribute 'not bad eh' numeric
@attribute 'not bad ley' numeric
@attribute 'not blurry' numeric
@attribute 'not blurry whataphone' numeric
@attribute 'not brilliant' numeric
@attribute 'not brilliant for' numeric
@attribute 'not camera' numeric
@attribute 'not camera front' numeric
@attribute 'not clear' numeric
@attribute 'not confidence' numeric
@attribute 'not confidence to' numeric
@attribute 'not edited' numeric
@attribute 'not edited and' numeric
@attribute 'not expect' numeric
@attribute 'not expect it' numeric
@attribute 'not giving' numeric
@attribute 'not giving it' numeric
@attribute 'not going' numeric
@attribute 'not going to' numeric
@attribute 'not last' numeric
@attribute 'not last enough' numeric
@attribute 'not look' numeric
```

```
@attribute 'not look at' numeric
@attribute 'not much' numeric
@attribute 'not much else' numeric
@attribute 'not much worse' numeric
@attribute 'not one' numeric
@attribute 'not one advert' numeric
@attribute 'not only' numeric
@attribute 'not only blackberry' numeric
@attribute 'not really' numeric
@attribute 'not really htc' numeric
@attribute 'not so' numeric
@attribute 'not so hot' numeric
@attribute 'not sure' numeric
@attribute 'not sure how' numeric
@attribute 'not sure i' numeric
@attribute 'not tech' numeric
@attribute 'not tech at' numeric
@attribute 'not to' numeric
@attribute 'not to buy' numeric
@attribute 'not too' numeric
@attribute 'not too favorable' numeric
@attribute 'not very' numeric
@attribute 'not very good' numeric
@attribute 'not working' numeric
@attribute 'not working good' numeric
```

We notice that some of the words of the above N-Grams have been reduced deliberately to their stem, base or root form. In the next section it is briefly explained why this is happening.

Stemmer

Stemming is a process where every word is reduced to its' stem. This stem does not need to be the morphological root of the word, but we want related words to map to the same stem.

We make use of the Iterated Lovins Stemming Algorithm. This is an improvement of the first ever published stemming algorithm, Lovins JB (1968). It is a relatively fast – single pass algorithm, which is able to handle efficiently grammatical effects like the removal of double letters in words (for example 'getting' is transformed to 'get'). Additionally it can handle many irregular plurals like – mouse and mice etc. In comparison to the Lovins Stemming algorithm it tries also to extenuate the over stemming and under stemming problem.

The combination of using N-Grams along with a stemmer helps us to deal with FR-4. That's because we do not only use a conventional lexicon that contains only formal and grammatically correct words. Contrariwise our algorithm is being trained to take advantage of every possible word combination regardless of whether it contains misspellings, abbreviations or slang language.

Use of Stoplist

It is not our intention to build an extremely large dataset. To moderate this problem we remove from our feature set very frequently used words or very short words (having less than two letters). These words appear almost equal number of times in all categories, so it makes no sense for someone to take them into account. The stopwords that were used are based on Rainbow. We used an english stopwords list, but it is relatively easy to add any stopword list we want.

4.1.3 Stacking Classifiers

As already mentioned our proposed solution combines two different classifiers using the stacking method that is briefly described in paragraph 2.3.2. A meta-classifier is then used to do the final classification taking into consideration the results of the other two classifiers.

The two classifiers that are "stacked" are:

SMOreg Classifier

SMOreg implements the support vector machine for regression.

PLS Classifier

PLSClassifier is a wrapper classifier for the PLSFilter. This classifier exploits the predictive ability of the PLSFilter, which runs Partial Least Square Regression over the given instances and computes the resulting beta matrix for prediction (Naes, Isaksson, Fearn, & Davies, 2002).

The Meta-classifier that is used by the Stacking method is the:

Kstar Classifier

K* is an instance-based classifier, which uses an entropy-based distance function (John G. Cleary, 1995).

4.2 Evaluation Criteria/Methodology

4.2.1 Training Data Set

As a training set we decided to use the same set of tweets that were used in (Kontopoulos, Berberidis, Dergiades, & Bassiliades, 2013). The tweets of this set are tweets containing keywords related to the domain of Mobile Phones. For all these tweets a preprocessing phase took place, which included removing strings that would not help in the training and the evaluation process. More precisely the following items were removed:

- URLs and tiny urls (which are used a lot in Twitter)
- Replies to other users' tweets (starting with '@')
- The "#" character that is used to indicate that a specific string is a hashtag.

This way we reduced some of the noise of our training data set. This initial dataset was consisted of 632 tweets (a lot of these were the same since they were automatically retrieved and many of these were retweets of other tweets). The next step was to remove all these duplicate tweets. We would not want in any case the same tweet to appear in the training set five or ten times for example, because this would lead to misleading inferences.

Most of the abovementioned tweets referred to only one aspect, but some of them referred to two or three different aspects. So the final step of preprocessing the data was to break them into sentence segments that refer to only one aspect. For example the tweet 'Hate my iPhone now = = Samsung-Galaxy SII Camera so damn clear' would lead to two distinct instances ('Hate my iPhone now' and 'Samsung-Galaxy SII Camera so damn clear'). Another example would be that of the following tweet:

'My phone problems were settled yesterday I now have an HTC-One X Quad core processor huge clear screen amazing camera crap battery life'

This breaks into five different segments that will be used for training.

1. 'My phone problems were settled yesterday'
2. 'I now have an HTC-One X Quad core processor'
3. 'huge clear screen'
4. 'amazing camera'
5. 'crap battery life'.

This way we ended up having 706 instances out of the initial 632 tweets. Of course if we used a dataset ten times this one, our classifier would have even better results.

Every instance has two attributes. The first attribute refers to the tweet itself (or the part of the tweet that refers to a certain aspect). Its' type is "string". Instances have been selected in a way that this attribute is unique in a percentage of 95%. To achieve this we simply removed from our dataset tweets with the same content.

The second attribute is the "Rating" and it is of type "Numeric". Its' values range one to six according to Functional Requirement FR-1. The rating's mean is 4.483 and the standard deviation is 1.624. Ideally we would prefer a mean around 3.5 and to have almost equal number of instances with negative, neutral and positive sentiment rating. This would have as a result for our model to have a lot better results but it would not be so realistic, as it would be something artificially built.

How exactly the 706 instances are divided in a scale of one to six is shown graphically below in figure 5.

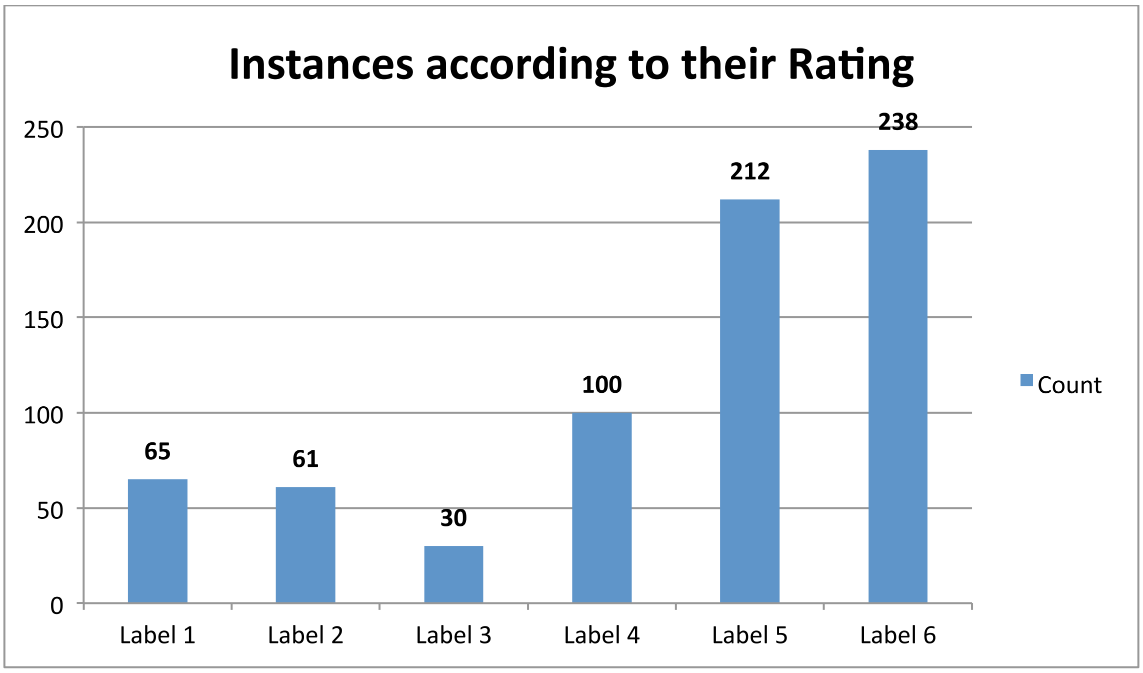


Figure 8 - Instances according to their Rating

4.2.2 Cross Validation and Evaluation Measures

What we wanted to ensure is that our experimental results would be the same as if they were obtained on completely independent test sets. To do so we used an evaluation technique called 10-fold Cross Validation, which practically does the following:

1. Break data into 10 sets of size n/10.
2. Train on 9 datasets and test on 1.
3. Repeat 10 times and take a mean accuracy.

We used 10-fold cross validation instead of 3-fold or 20-fold, because extensive tests on many different datasets have shown that with 10-fold cross validation is what is needed to get the best estimate of error.

Another way to do so would be to use a large training set in order to build our model and then test it on another large independent test set. Of course in this case we suppose that both samples are representative.

One big decision that we had to take was the choice of the right evaluation metrics for the comparison of the various algorithms that were used.

We already mentioned one of such metrics when speaking of the Nominal Classifiers in paragraph 4.1.1, which is the percentage of correctly classified instances. But in the end we decided that it is preferable to work with regression algorithms. Error rates are used for numeric prediction. In numeric prediction, predictions aren't just right or wrong, the error has a magnitude, and these measures reflect that. There are many performance measures for evaluating numeric prediction as in our case. We decided to use Mean-absolute error and Relative-absolute error, which we considered to be most appropriate and easier to understand. We will explain briefly each of these two metrics.

Consider that the predicted values of the instances of the training test are pr_1, pr_2, …, pr_n, whereas the actual values are ac_1, ac_2, …, ac_n and $\overline{ac}$ is the mean value over the training data.

Mean-absolute error is defined by the following formula:

$$\text{Mean} - \text{absolute error} = \frac{|\text{pr1} - \text{ac1}| + \cdots + |\text{prn} - \text{acn}|}{\text{n}}$$

Relative-absolute error is defined by:

$$\text{Relative} - \text{absolute error} = \frac{|\text{pr1} - \text{ac1}| + \cdots + |\text{prn} - \text{acn}|}{|\text{ac1} - \overline{ac}| + \cdots + |\text{acn} - \overline{ac}|}$$

Therefore, **smaller values** for both these measures **are better.**

The results of the experiments are shown in section 4.3.

Something else that played very important role to our final choice of the classification scheme that was used was the ability that is offered to us by weka to output all the predictions of a regression algorithm for the instances that were used as a training set.

4.3 Experimental Results

In this point we should mention that the data that were used for the training and the evaluation of the models are:

1. seperated_tweets_OneToSix – Nominal.arff
2. seperated_tweets_OneToSix – Numeric.arff

Below are presented only few of the experimental results.

The results below are taken after having decided to work with regression algorithms, N-Grams and the Iterated Lovins Stemmer.

In Figure 9 is displayed the performance of seven different regression algorithms. We observe that SMOReg and PLSClassifier have the best performance among the others.

Classifier used	**Mean Absolute Error**	**Relative Absolute Error**
SimpleLinearRegression	1.1965	89.5046 %
SMOReg	**0.92**	**69.17%**
PLSClassifier	**0.88**	**65.92%**
Gaussian Processes for regression without hyperparameter-tuning	1.2945	96.8355 %
K-nearest neighbours classifier (IBk)	1.1843	88.5923 %
Using a decision stump	1.1965	89.5046 %
M5Base (Implements base routines for generating M5 Model trees and rules)	1.0502	78.5596 %

Figure 9 - Comparing Regression Algorithms after having chosen the right filters

In the next figure we compare the previous two best algorithms (SMOReg and PLSClassifier) with two ensemble techniques (Stacking and Vote) that combine these two classifiers.

Classifier used	Mean Absolute Error	Relative Absolute Error
Stacking of SMOreg, PLSClassifier with KStar as meta classifier	**0.88**	**67%**
SMOReg	0.92	69.17%
PLSClassifier	0.88	65.92%
Vote using SMOreg, PLSClassifier, KStar and Maximum Probability	0.90	67%
Vote using SMOreg, PLSClassifier, KStar and Minimum Probability	0.90	67%

Figure 10 - Comparing the 2 best Regression Algorithms with Ensemble Methods

As we can see from the above figure the difference by just observing the measures is not significant. The difference resides in the fact that the predicted values using the Stacking Method are much closer to the actual ones compared to any other method. We can see the result of output predictions using the **Stacking of SMOreg and PLSClassifier with KStar** as meta classifier in the next frame for some of the training instances.

```
=== Predictions on test data ===

  inst#,    actual, predicted, error

     1       4          5.105      1.105
     2       6          4.225     -1.775
     3       5          4.84      -0.16
     4       6          4.047     -1.953
     5       6          5.673     -0.327
     6       4          4.102      0.102
     7       4          5.008      1.008
     8       6          5.307     -0.693
     9       5          4.324     -0.676
    10       6          5.464     -0.536
    11       2          3.773      1.773
    12       2          3.105      1.105
    13       5          5.288      0.288
    14       6          4.174     -1.826
    15       4          4.913      0.913
```

16	4	4.768	0.768
17	1	2.299	1.299
18	5	5.142	0.142
19	5	3.149	-1.851
20	6	5.231	-0.769
21	4	3.37	-0.63
22	5	5.028	0.028
23	3	4.642	1.642
24	4	3.928	-0.072
25	6	5.722	-0.278
26	6	5.455	-0.545
27	5	4.854	-0.146
28	1	3.227	2.227
29	4	3.772	-0.228
30	6	5.194	-0.806
31	5	3.953	-1.047
32	6	5.469	-0.531
33	5	3.686	-1.314
34	5	4.891	-0.109
35	2	3.114	1.114
36	4	5.174	1.174
37	4	3.642	-0.358
38	5	4.598	-0.402
39	6	5.752	-0.248
40	5	4.828	-0.172
41	3	5.056	2.056
42	4	5.082	1.082
43	6	5.147	-0.853
44	4	4.885	0.885
45	2	4.002	2.002
46	1	2.178	1.178
47	5	4.665	-0.335
48	3	4.062	1.062
49	2	2.566	0.566
50	6	5.76	-0.24
51	1	3.957	2.957
52	5	5.078	0.078
53	4	4.032	0.032
54	2	3.116	1.116
55	3	3.111	0.111
56	5	5.533	0.533
57	4	3.853	-0.147
58	5	5.205	0.205
59	6	5.255	-0.745
60	4	5.014	1.014
61	4	2.896	-1.104
62	3	1.853	-1.147
63	2	1.995	-0.005

```
    64       5           4.231      -0.769
    65       5           4.776      -0.224
    66       2           3.13        1.13
    67       6           5.13       -0.87
    68       6           5.724      -0.276
    69       2           3.16        1.16
    70       2           3.797       1.797
```

We see that with Stacking we don't simply get a mean absolute error of 0.88, which is extremely significant but we don't have many outliers. This means that we don't see instances that should be rated as “two” to get a prediction of “five” or “six”.

Finally another reason that we prefer the Stacking over the Vote method is that it is much faster.

5 Conclusions and Future Work

One of the biggest advantages that our approach offers is that it is completely independent of the language that the documents we want to rate are written in. This algorithm can be used to create models for giving a basic sentiment rating in any language. The reason is that no lexicon is being used, only N-Grams from the training documents that are given to the algorithm as an input.

Something that would be very useful and relatively simple as a future improvement is to find a way to indentify the language in which a document is written and use in some automatic way the appropriate model in order to rate it. To classify documents according to their language we could use N-Gram Frequency Profiles (Cavnar & John, 1994).

The most difficult part that should be done, in order to speak of a complete sentiment analysis tool is to create the mechanism that will indentify all the entities of a document. After doing so, the same mechanism should break the sentences in appropriate parts. These parts will be given to our algorithm as arguments pending for rating. Then according to the ontology that each entity belongs to, the appropriate model will be chosen to be used for the evaluation.

Bibliography

Antoniou, G. C. (2012). *Creation of a Twitter web-hint system that propose tweets based on the user preferences.* Thessaloniki, Greece: Dept. of Electrical and Computer Engineering, Aristotle University of Thessaloniki .

B.V., B. M. (2013). *Open Dover*. (Byelex Multimedia Products B.V.) Retrieved Oktober 2013, from http://www.opendover.nl/

Bakliwal, A., Foster, J., van der Puil, J., O'Brien, R., Tounsi, L., & Hughes, M. (2013). Sentiment Analysis of Political Tweets: Towards an Accurate Classifier. *Proceedings of the Workshop on Language in Social Media (LASM 2013)*, (pp. 49-58). Atlanta, Georgia.

Barbosa, L., & Feng, J. (2010, August). Robust Sentiment Detection on Twitter from Biased and Noisy Data. *Coling 2010: Poster Volume*, pp. 36-44.

Brody, S., & Elhadad, N. (2010). An Unsupervised Aspect-Sentiment Model for Online Reviews. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL* (pp. 804-812). Los Angeles: Association for Computational Linguistics.

Cavnar, W. B., & John, T. M. (1994). N-Gram-Based Text Categorization. *Environmental Research Institute of Michigan*.

Cerini, S., Compagnoni, V., Demontis, A., Formentelli, M., & Gandini, G. (2007). Micro-WNOp: A gold standard for the evaluation of automatically compiled lexical resources for opinion mining. *Andrea Sansò, ed., Language Resources and Linguistic Theory: Typology, Second Language Acquisition, English Linguistics*.

Cianciullo, J. (n.d.). *SocialMention*. Retrieved Oktober 2013, from http://www.socialmention.com/

Davidov, D., Tsur, O., & Rappoport, A. (2010, August). Enhanced Sentiment Learning Using Twitter Hashtags and Smileys. *Coling 2010: Poster Volume*, pp. 241-249.

Dergiades, T. (2012). Do investors’ sentiment dynamics affect stock returns? Evidence from the US economy. *Economics Letters, 116(3)*, pp. 404-407.

Esuli, A., & Sebastiani, F. (2006). SENTIWORDNET: A Publicly Available Lexical Resource.

Ganter, B. B., & Wille, R. (1999). *Formal concept analysis, mathematical foundation.* Berlin: Springer Verlag.

Gruber, T. R. (1993, June). A translation approach to portable ontology specifications. *Knowledge Acquisition 5 (2)*, pp. 199-220.

Hall, M., & Frank, E. (2001). A Simple Approach to Ordinal Classification. *12th European Conference on Machine Learning*, (pp. 145-156).

Hu, M., & Liu, B. (2004). Mining and summarizing customer reviews. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 168–177).

John G. Cleary, L. E. (1995). K*: An Instance-based Learner Using an Entropic Distance Measure. *12th International Conference on Machine Learning*, (pp. 108-114).

Kontopoulos, E., Berberidis, C., Dergiades, T., & Bassiliades, N. (2013). Ontology-based sentiment analysis of twitter posts. *Elsevier*(Expert Systems with Applications 40 (2013)), pp. 4065–4074.

Kouloumpis, E., Wilson, T., & Moore, J. (2011). Twitter Sentiment Analysis: The Good the Bad and the OMG! *Association for the Advancement of Artificial.*

Kumar, S., Morstatter, F., & Liu, H. (2013). *Twitter Data Analytics.* Springer.

Liu, B. (2012). Sentiment Analysis and Opinion Mining. *AAAI-2011, EACL-2012, and Sentiment Analysis Symposium.*

Liu, Y., Huang, X., An, A., & Yu, X. (2007). *ARSA: A Sentiment-Aware Model for Predicting Sales* (Vol. SIGIR’07). Amsterdam.

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM 38(11)*, pp. 39–41.

Naes, T., Isaksson, T., Fearn, T., & Davies, T. (2002). *A User Friendly Guide to Multivariate Calibration and Classification.* NIR Publications.

Nebhi, K. (2012). Ontology-Based Information Extraction from Twitter. *Proceedings of the Workshop on Information Extraction and Entity Analytics on Social Media Data* (pp. 17-22). Mumbai: COLING.

Pang, B., & Lee, L. (2004). A Sentimental Education: Sentiment Analysis Using Subjectivity. *ACL '04*, (pp. 271–278).

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning. *Proceedings of EMNLP*, (pp. pp. 79–86).

Park, S., Ko, M., Kim, J., Liu, Y., & Song, J. (2011). The Politics of Comments: Predicting Political Orientation. *ACM 978-1-4503-0556-3/11/03.* Hangzhou.

Stanford University. (2013). *The Stanford Natural Language Processing Group*. Retrieved from http://nlp.stanford.edu/software/corenlp.shtml

Stone, P. J., Dunphry, D., Smith, M., & Ogilvie, D. (1966). *The General Inquirer: A Computer Approach to Content Analysis.* Cambridge: MIT Press.

Tsoumakas, G., & Vlahavas, I. (n.d.). *Effective Stacking of Distributed Classifiers.* Thessaloniki: Dept. of Informatics, Aristotle University of Thessaloniki.

Tweet Sentiment Visualization. (n.d.). Retrieved Oktober 2013, from Sentiment Viz: http://www.csc.ncsu.edu/faculty/healey/tweet_viz/tweet_app/

Waikato, D. o. (2013, Oktober). *Weka 3: Data Mining Software in Java*. Retrieved from http://www.cs.waikato.ac.nz/~ml/weka/

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining, Practical Machine Learning Tools and Techniques* (3rd ed.). U.S.A.: Morgan Kaufmann Publishers.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., . . . Steinberg, D. (2007). Top 10 algorithms in data mining. *Springer-Verlag*.

Zhang, L., Ghosh, R., Dekhil, M., Hsu, M., & Liu, B. (2011). Combining Lexicon-based and Learning-based Methods for Twitter. *HP Laboratories*(HPL-2011-89).

Zhang, L., Xu, W., & Li, S. (2012). Aspect identification and sentiment analysis based on NLP. *Network Infrastructure and Digital Content (IC-NIDC), 2012 3rd IEEE International Conference on*, (pp. 660 - 664). Beijing.

Appendix

Below is presented the source code for the main class of the MessageClassifier. It is based on a sample application that is used in a companion book for the Weka software, which is written by the development team of Weka (Witten, Frank, & Hall, 2011). We could consider it as a manual that describes various machine learning techniques and how these techniques can be used with the help of Weka. There are some comments that make the reading of the code easier.

```
package SentimentAnalysis;

/**
 *
 * @author Antoniou Georgios
 */
/**
* Java program for attaching a sentiment score from one to six.
*/
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.Serializable;
import weka.classifiers.Classifier;
import weka.classifiers.functions.PLSClassifier;
import weka.classifiers.functions.SMOreg;
import weka.classifiers.lazy.KStar;
import weka.classifiers.meta.FilteredClassifier;
import weka.classifiers.meta.Stacking;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.SerializationHelper;
import weka.core.Utils;
import weka.core.stemmers.IteratedLovinsStemmer;
import weka.core.tokenizers.NGramTokenizer;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.StringToWordVector;


public class MessageClassifier implements Serializable {
```

```
/** The training data gathered so far. */
private Instances m_Data = null;
/** The filter used to generate the word counts. */
private StringToWordVector m_Filter = new StringToWordVector();
private IteratedLovinsStemmer Stem = new IteratedLovinsStemmer();
private NGramTokenizer NGram = new NGramTokenizer();
/** The actual classifier. */
private FilteredClassifier m_Classifier = new FilteredClassifier();
private Stacking m_Classifier1 = new Stacking();
private Classifier m_Classifier2 = new SMOreg();
private Classifier m_Classifier3 = new PLSClassifier();
private    Classifier[]    StackingClassifiers    =    {m_Classifier2,
m_Classifier3};
private Classifier m_Classifier4 = new KStar();

/** Whether the model is up to date. */
private boolean m_UpToDate;
/** For serialization. */
private static final long serialVersionUID = -123455813150452885L;
/**
* Constructs empty training dataset.
*/
public MessageClassifier() {
String nameOfDataset = "MessageClassificationProblem";
String Rangelist = "first";
// Create vector of attributes.
FastVector attributes = new FastVector(2);
// Add attribute for holding messages.
attributes.addElement(new Attribute("Message", (FastVector) null));
// Add class attribute which is numeric.
attributes.addElement(new Attribute("Class"));
//  Create  dataset  with  initial  capacity  of  100,  and  set  index  of
class.
m_Data = new Instances(nameOfDataset, attributes, 100);
m_Data.setClassIndex(m_Data.numAttributes() - 1);
m_Classifier1.setClassifiers(StackingClassifiers);
m_Classifier1.setMetaClassifier(m_Classifier4);

m_Filter.setAttributeIndices(Rangelist);
m_Filter.setDoNotOperateOnPerClassBasis(true);
m_Filter.setInvertSelection(false);
m_Filter.setLowerCaseTokens(true);
m_Filter.setMinTermFreq(1);
// We are interested only in which terms occur and not how many times
each one occurs.
m_Filter.setOutputWordCounts(false);
m_Filter.setStemmer(Stem);
m_Filter.setUseStoplist(true);
m_Filter.setTokenizer(NGram);
m_Filter.setWordsToKeep(10000000);
m_Classifier.setFilter(m_Filter);
m_Classifier.setClassifier(m_Classifier1);


}
/**
* Updates model using the given training message.
*
* @param message the message content
* @param classValue the class label
*/
public void updateData(String message, int classValue) {
// Make message into instance.
Instance instance = makeInstance(message, m_Data);
// Set class value for instance.
instance.setClassValue(classValue);
// Add instance to training data.
m_Data.add(instance);
m_UpToDate = false;
}
/**
* Classifies a given message.
*
* @param message the message content
* @throws Exception if classification fails
*/
public void classifyMessage(String message) throws Exception {
// Check whether classifier has been built.
if (m_Data.numInstances() == 0) {
throw new Exception("No classifier available.");
}
// Check whether classifier and filter are up to date.
if (!m_UpToDate) {
// Initialize filter and tell it about the input format.
m_Filter.setInputFormat(m_Data);
// Generate word counts from the training data.
//Instances filteredData = Filter.useFilter(m_Data, m_Filter);
// Rebuild classifier.
m_Classifier.buildClassifier(m_Data);
m_UpToDate = true;
}
// Make separate little test set so that message
// does not get added to string attribute in m_Data.
Instances testset = m_Data.stringFreeStructure();
// Make message into test instance.
Instance instance = makeInstance(message, testset);
// Filter instance.
```

```
m_Filter.input(instance);
Instance filteredInstance = m_Filter.output();
// Get index of predicted class value.
double predicted = m_Classifier.classifyInstance(filteredInstance);
// Output class value.
System.err.println("Message classified as : " +
m_Data.classAttribute().value((int) predicted));
}
/**
* Method that converts a text message into an instance.
*
* @param text the message content to convert
* @param data the header information
* @return the generated Instance
*/
private Instance makeInstance(String text, Instances data) {
// Create instance of length two.
Instance instance = new Instance(2);
// Set value for message attribute
Attribute messageAtt = data.attribute("Message");
instance.setValue(messageAtt, messageAtt.addStringValue(text));
// Give instance access to attribute information from the dataset.
instance.setDataset(data);
return instance;
}
/**
* Main method. The following parameters are recognized:
*
* -m messagefile
* Points to the file containing the message to classify or use
* for updating the model.
* -c classlabel
* The class label of the message if model is to be updated.
* Omit for classification of a message.
* -t modelfile
* The file containing the model. If it doesn't exist, it will
* be created automatically.
*
* @param args the commandline options
*/
public static void main(String[] args) {
try {
// Read message file into string.
String messageName = Utils.getOption('m', args);
if (messageName.length() == 0) {
throw new Exception("Must provide name of message + file ('-m
<file>').");
}
FileReader m = new FileReader(messageName);
StringBuffer message = new StringBuffer();
int l;
while ((l = m.read()) != -1) {
message.append((char) l);
}
m.close();
// Check if class value is given.
String classValue = Utils.getOption('c', args);
int classValue2 = 0;
if (classValue.length() != 0) {
classValue2 = Integer.parseInt(classValue);
}
// If model file exists, read it, otherwise create new one.
String modelName = Utils.getOption('t', args);
if (modelName.length() == 0) {
throw   new   Exception("Must   provide   name   of   model   +   file   ('-t
<file>').");
}
MessageClassifier messageCl;
try {
messageCl =(MessageClassifier) SerializationHelper. read(modelName);
} catch (FileNotFoundException e) {
messageCl = new MessageClassifier();
}
// Check if there are any options left
Utils.checkForRemainingOptions(args);
// Process message.
if (classValue.length() != 0) {
messageCl.updateData(message.toString(), classValue2);
} else {
messageCl.classifyMessage(message.toString());
}
// Save message classifier object only if it was updated.
if (classValue.length() != 0) {
SerializationHelper.write(modelName, messageCl);
}
} catch (Exception e) {
e.printStackTrace();
}
}
}
```