SMME

Social Media Monitoring Engine

Student: Lee Kerby
Registration: 0838278

Supervisor: Dr. Udo Kruschwitz

Abstract

Monitoring social trends of the population has recently experienced an explosion of interest from sociology to business economics. Data mining into opinion polarity is a forerunner of research encapsulating this interest. Computer science desires to take this a step further by processing text using automated means; this is the aim of the Social Media Monitoring Engine.

The principle behind this project is to create an automated system that can quickly analyse the trends and opinions of the public. This is accomplished by using a statistical/rule-base hybrid sentiment engine to process text and determine if it contains a positive or negative opinion. Twitter has been used as the social media source; however, any repository can easily be integrated into SMME due to the concept of automated corpus generation.

In terms of research, SMME has exposed new areas of natural language engineering. A new classifier using statistical data and syntactical rule-sets has been designed and coded. The introduction of such a classifier could potentially propose new principles in NLE and opinion gathering (even if negative). SMME also includes the design of a new system to automate the generation of a corpus which when combined with the new classifier has produced accuracy results of 71% on only a 10,000 tweet corpus.

(Kerby, SMME: Abstract, 2011)

Table of Contents

1. Glossary of Terms

Term	Definition
SMME	Social Media Monitoring Engine
NLE	Natural Language Engineering
Twitter	A Social Media platform to read and write text
Tweet	140 Character message on the Twitter platform
Tweeted	The act of creating a Tweet
JSP	Java Server Pages
JSON	JavaScript Object Notation
Classifier	Part of the sentiment engine used to assess (classify) text
Corpus	Set of data a statistical classifier is trained on
SA	Sentiment Analysis
Bigram	One word
Unigram	Two words in succession
Trigram	Three words in succession
POS	Part of Speech (Tagging). Tags each token (word) into its lexical semantic group (noun, verb, adjective etc).
Tokenizer	Splits text into tokens.
Tokens	A token is an individual word or character of importance.

2. Introduction

An introduction to SMME and Sentiment Analysis.

Social media incorporates the collective minds of the internet, a repository of opinionated human text. This information has unmeasured potential in the fields of product development such as awareness, tracking, competition comparison, discovery of desirables, regional proof and current trends. Capturing this information autonomously and presenting it in a useable manner would not only gain an insight for the many different academic fields and personal use but also provide an advantage to commercial industries. The intention of creating SMME is to report current and changing opinions of items, places and people using social media as its source of information (Kerby, SMME: Interim Report).

It is estimated the social media platform Twitter retrieves up to 177 million tweets per day (Penner, 2011). Considering the quantity and media being evaluated, Twitter is an excellent repository for sentiment analysis. Twitter is an embodiment of objective and subjective human statements on almost any given subject and as such, is a perfect platform for SMME to be trialed on. Using this information SMME is able to make relevant assessments on up to date and real time data.

Sentiment analysis essentially means to automate the judgment if a given statement is positive or negative in reference to the input. The intention of SMME is to accurately perform sentiment analysis on a given input using Twitter as its source of data. This is a difficult research domain due to the natural ambiguity of text and because computers have no pre-defined knowledge or intelligence in regards to human linguistic context.

Example:

Tweet:	*"I really like my cat"*
Term:	*"cat"*
Output:	*SMME should be able to determine that the statement is positive.*

SMME accepts a term, extracts up to 7,500 tweets from twitter over the past five days relating to that term, performs sentiment analysis on them and presents a report. The report displays the average sentiment, the positive and negative reviews each day and also commonly re-occurring related keywords.

The aim of SMME is not to simply research methods on how sentiment analysis is possible and mimic them, but instead develop a new system not currently used in the NLE field. This produces very interesting results and tends to push the project more into a research field rather than as a commercial product.

3. Related Work

The chapter produces an overview on influential work, key research studies and a look at Twitter SA competitors.

In terms of project development two distinct stages required exhaustive procurement of information, data mining of twitter and the performing of sentiment analysis. As expected, the convergence of ideas in how SMME was to be made escalated from related work.

Sentiment analysis is an active field amongst NLE producing new creations and theories in rapid progression each year, this is evident in the multiple institutions (York, Stanford Etc.) and companies (IBM, Google Etc.) all intent on discovering new techniques for both academic and profitable gain. Coupling this with data mining the automation of analysing text has unmeasured potential. Considering SMME is an experimental research project attempting to design and implement a new form of sentiment analysis engine lots of time was invested into current twitter sentiment engines, different sentiment analysis techniques and papers on proceedings.

3.1 Commercial Social Media Sentiment Analysis Engines

"product development such as awareness, tracking, competition comparison, discovery of desirables, regional proof and current trends". (Kerby, SMME: Interim Report)

The process of providing a service which data mines people's opinions from a source and displays the results is very lucrative. As a result there are many companies offering this as a paid product (RavenPack, SproutSocial, Telligent, Synaptrics, Sentiment Metrics). Exactly how these companies systems work is a trade secret, but it does show the potential and plausibility of SMME as a commercial product.

Further research into competitors show that there is a large amount of Twitter specific free web based sentiment engines available. Analysing how these work provided a backbone on what SMME should be aiming for and what is currently being developed in a similar low budget low resources environment.

Twitter sentiment analysis engine comparison
Table 1

The table below compares the variation in free Twitter sentiment analysis engines.

Product	Website	Sentiment Engine
Twitter Search	http://search.twitter.com/	:) and :(icons used inside a Tweet
Twitter Sentiment	http://twittersentiment.appspot.com/	Naïve Bayes & Corpus
Twitrratr	http://twitrratr.com/	Analyses the search term against a list of positive and negative keywords
Twendz	http://twendz.waggeneredstrom.com/	Combination of keywords and symbols which are then cross-reference against a list of positive and negative keywords

(Search Engine Journal, 2010) – Data confirmed on the 1st April 2011.

The different types of sentiment engines available to the public use two distinct variations for processing sentiment.

- Rules
- Statistical

From the above examples the "Twitter Sentiment" is statistical, the rest are rule based. There is no clear indication of accuracy for any of these sites however the statistical engine boasts an accuracy rating of 80 percent and published by a team at Stanford University. Much procrastination on their site guesstimated that this percentage is slight fallacy, yet it has greater accuracy than the others and has the ability to expand as it's based on Bayes Theorem. SMME is designed to not rival Bayes Theorem but instead provide an alternative solution to statistical sentiment analysis that could be considered a success when comparing the resources spent and accuracy gained.

The data mining used by these competitors is closely guarded. Emails were distributed to all parties however only one reply was received from Alec Go, the co-creator of the statistical classifier "Twitter Sentiment" expressing the use of the Twitter streaming API. Unfortunately such an API is irrelevant for SMME, as it requires a history of Tweets that isn't provided by a stream.

3.2 Statistical Sentiment Analysis Research

Research into the competitors helped build the conception of SMME's underlying engine. It encouraged the exploration of original ideas and instigated the use of a statistical classifier. This was primarily due to the aspirations of a challenging (but rewarding) project and the advantages such classifier delivers. Multiple papers within the NLE fields were scanned for any related work to gather some ideas. Three papers resonated fundamental knowledge that became consistent reference material throughout development.

A Sentiment Analysis Model Integrating Multiple Algorithms and Diverse Features
(Xu, 2010)
A thesis written with the intention to summaries the different common algorithms used for sentiment analysis. It covers a broad spectrum of techniques and hinted towards multiple different directions a classifier can be built around and estimates on their accuracy. This thesis provoked the interest in a PMI model defined by Dr. Turney (see 'Important Relevant Work')

Thumbs up? Sentiment Classification using Machine Learning Techniques
(Pang, Lee, & Vaithyanathan, 2002)
The first paper of many which is relevant to SMME. It explains how Naïve Bayes Theorem, Maximum Entropy and Support Vector Machines can be used to determine polarity within movie reviews. The paper revolves around the use of a statistical approach and mentions how corpus generation needs to be generated manually. Interesting facts emerged showing the potential of Bayes Theorem producing results in the 80 percent range.

Twitter as a Corpus for Sentiment Analysis and Opinion Mining
(Pak & Paroubek, 2010)
An interesting relevant study performed on sentiment analysis within the Twitter domain. It follows the trend of using Bayes Theorem but is the first to use a simplistic form of automated corpus generation. The paper also explicitly defines which lexical semantics are generally found within subjective and objective texts. Unfortunately this paper was discovered very late in the design and implementation cycle of SMME as it holds some very relevant studies that could be applied to this project.

3.3 Important Relevant Research

Twitter Sentiment Classification using Distant Supervision
(Go, Bhayani, & Huang, 2009)
The website sentiment engine described earlier as "Twitter Sentiment" revolves around the basis of this paper. Once again it utilises Bayes Theorem but hints at the prospect of generating a corpus automatically. It does this by extracting thousands of Tweets that contain an emoticon. It is then assumed the tweets with a ':)' are deemed positive and those with ':(' are negative for training

purposes which is the same method Pak & Paroubek use. As one can imagine this is not ideal (in terms of ambiguity) however these are the only papers in the field found which contains a non-manual corpus. This established the inspiration on what would make SMME unique.

Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews

(Turney, 2002)

Dr. Turney’s work in this paper implicitly defines the PMI method for sentiment analysis. It is not the SA method that is related to SMME, but rather a sub-study on opinionated statements. Multiple trigrams proven to be apparent in statements containing polarity are documented. This research is strongly related to SMME as it defined how SMME determines polarity in a statement when automating the process of corpus generation.

4 Requirements

The requirements defined for SMME, originated from the Interim Report.

4.1 Required

Functional Requirements:

- Allow queries on people, places or items.
- Retrieve information from social media repository dependent on the query given.
- Create a working polarity assessment engine that produces non-random results.
- The polarity assessments engine to use a method other than simple word polarity occurrence checking to work out sentiment.
- Seventy percent or greater accuracy in polarity assessment. This means, out of all the tweets processed with sentiment analysis, seventy percent or more must correctly be tagged with a positive, negative or neutral opinion.
- Present findings in a graphical report format.

Non-Functional Requirements:

- Achieve a turn-around speed of less than ten seconds.
- Host on a local server.
- Provide an interface according to web standards and effective UI.
- Code must have effective error handling without causing crashes.
- Allow concurrency between multiple users.

4.2 Desirable

Functional Requirements:

- Produce a seventy percent accuracy or greater in assessing context ambiguity. This means, out of all the tweets processed for polarity, seventy percent or more of tweets that contain context ambiguity must still output the correct polarity.
- Display changes of polarity over the course of a chronological timeline.
- All processed tweets that have geographic data attached to their profiles the polarity is displayed appropriately on a map.
- All tweets that have gender data attached to their profiles are reported appropriately.
- Report reoccurring discussed topics within the given search query
    - *IE: Search query "Xbox 360" may return results "RROD, High Definition, Halo, GTA"* [12].

Non-Functional Requirements:

- Host on a server to be used from any location.

5. Design

This chapter implicitly defines the current design schematics for SMME. It clearly indicates how and why SMME processes sentiment analysis.

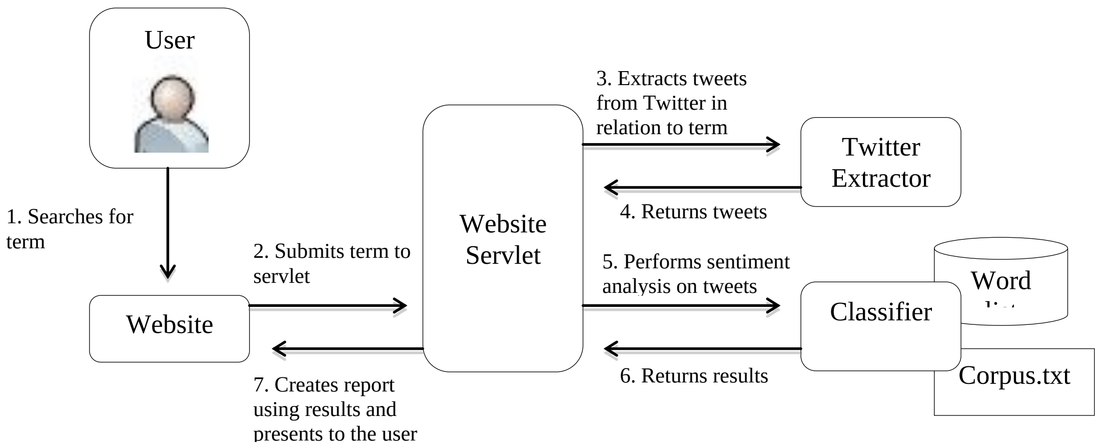
Mimicking the designs of a pre-existing sentiment analyses engine seemed nonsensical in the domain of an individual project. As a result the intention of SMME is designed to be original. This has lead to the development of an original classifier not currently used in the NLE field for sentiment processing and an automatically generating corpus system. Aside from the classifier and corpus, SMME requires several distinct steps to achieve its requirements. The design of SMME is broken down into several stages for modularity and efficiency. The classifier and corpus generator are the most prestigious and complex and naturally command the most attention in this report.

5. 1 Modular parts of SMME:

- **Classifier**
  - *Processes the tweets polarity in reference to the user inputted term.*
- **Corpus Generator**
  - *Generates the 'knowledge' for the classifier, what is an opinionated tweet.*
- **Twitter Extractor**
  - *Extracts tweets from Twitter for each search and corpus generation.*
- **Word list**
  - *A list of words and their polarity.*
- **Website**
  - *Allows interaction for the user to search for a term and also view the report.*

Diagram of interaction between modules:
Figure 1

The user inputs a term on the website that is forwarded to the servlet. The servlet uses the twitter extractor to acquire up to 7,500 tweets from the past five days and forwards them onto the classifier. The classifier performs sentiment analysis on the tweets. The servlet creates a report using this data and redirects the user.


User
1. Searches for term
Website
2. Submits term to servlet
7. Creates report using results and presents to the user
Website Servlet
3. Extracts tweets from Twitter in relation to term
4. Returns tweets
Twitter Extractor
5. Performs sentiment analysis on tweets
6. Returns results
Classifier
Word list
Corpus.txt

5.2 Classifier

The classifier is the most independent important feature of this project. It alone is a significant dividing unique feature and enables the system to produce automated sentiment analysis. The classifier for SMME enables a tweet to be assessed for a positive, negative or neutral polarity.

As briefly described in related work section there are already an abundance of popular classifier methods. The most endorsed solution sporting a high accuracy is the statistical method Naïve Bayes. Naïve Bayes Theorem uses the 'bag of words' approach. There is no syntactic analysis knowledge used in this method, instead probability of each cluster (unigram, bigram or trigram) of words are evaluated. It uses a manually defined corpus, which it requires to 'learn' from.

The positives and negatives of using Naïve Bayes:
Table 2

Positive	Negative
Modularity: The use of a corpus means that it can be trained on any data. For example, a change of language or syntactical structure.	Manual Corpus: The corpus needs to be manually constructed by a human. Naïve Bayes is more accurate with a bigger corpus; some corpuses are in the millions of words.
Accuracy & Efficiency: All research papers suggest the statistical approach is better due to the time saved on created an endless and mutable list of rules.	Ambiguity: Such as all NLE methods to date, solving ambiguity in text is almost impossible. This is extenuated in Naïve Bayes as it has no syntactical or context knowledge of text.

On the other spectrum of NLE sentiment analysis is the rule-based classifier. This is often adopted due to its simplicity of production. There are multiple variations of this but all focus on the syntactical structure of a sentence and perform a list of rules. As one can expect the modularity is limited and the expansion is restricted but it is quick to develop.

The design for SMME classifier is a hybrid of both the statistical and rule based approaches. The theory combines the effectiveness of having a learning algorithm (statistical) to comprehend the main polarity detection and the rules for tweaking performance. The designs on how this works are completely different to Naïve Bayes. The reason for this is to keep the originality, push into different realms of NLE research and also try and correct the flaws that Naïve Bayes has.

The SMME classifier searches for the word inside the sentence that is the main indication of polarity. Naïve Bayes looks at the probability of all words occurring inside a positive, negative or neutral sentence. **This is the most important dividing feature.** The SMME classifier is more focused on syntactical relations between words rather than probability of word occurrence. This makes SMME very significant in the NLE domain as it uses statistical **syntactical** knowledge, something not used in current statistical classifiers. SMME tries to understand context of a syntactical structure rather than blindly applying probability.

The SMME classifier utilises a corpus much like all statistical engines with the addition of rule integration. The corpus it uses is a list of trigrams and their occurrence rate and is discussed further under “Automated Corpus Generation”. The trigrams train the classifier on where the word containing polarity is inside the tweet in regards to the term being searched.

5.2.2 Sentiment Processing

The classifier processes sentiment by looking for trigrams inside text that are known to contain polarity in relation to the input keyword, the trigrams being searched for are defined inside the corpus. Once a trigram is found, the classifier compares the words inside the trigram to a wordlist. The wordlist as described later contains words and their polarity (positive or negative). If a word is matched inside the wordlist, the polarity of the word is assigned to the text. Assuming there are multiple trigrams found, the most popular trigram (inside the corpus) is used. If a trigram is found which contains the KEY tag (the term searched for), the probability of that trigram being used for determining polarity is multiplied.

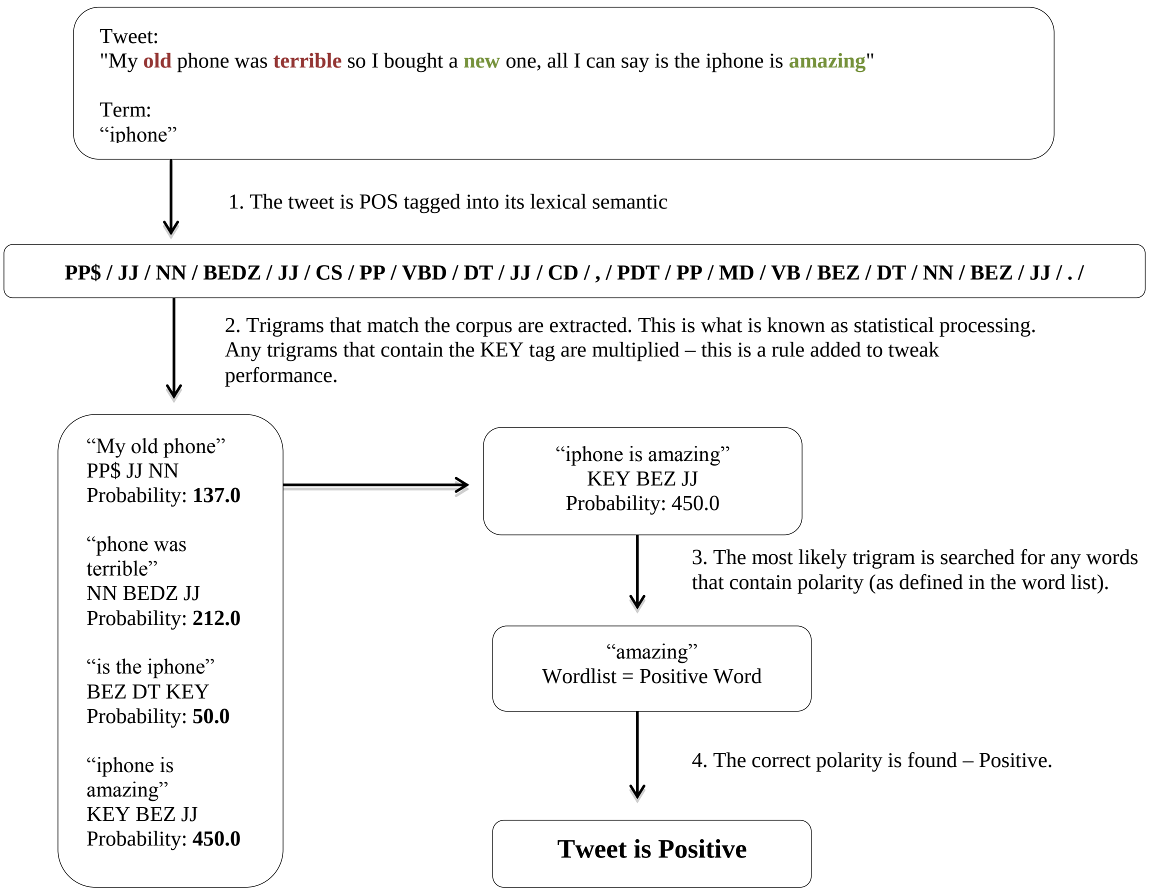
<u>Diagram of SMME classifier processing example:</u>
Figure 2

Corpus

Trigram	**Occurrence**
[JJ NN IN]	588.0
[RB JJ NN]	380.0
[JJ JJ NN]	320.0
[JJ NN NN]	314.0
[NN JJ NN]	310.0
[RB JJ IN]	266.0
[JJ NN ,]	252.0
[BEZ RB JJ]	246.0
[DT JJ KEY]	233.0

….etc.

This is a difficult statement to process as there are four words all containing some form of polarity. SMME uses the corpus for statistical processing to determine which word contains the polarity of the tweet in regards to the term “iphone”.



Tweet:
"My **old** phone was **terrible** so I bought a **new** one, all I can say is the iphone is **amazing**"

Term:
“iphone”

1. The tweet is POS tagged into its lexical semantic

PP$ / JJ / NN / BEDZ / JJ / CS / PP / VBD / DT / JJ / CD / , / PDT / PP / MD / VB / BEZ / DT / NN / BEZ / JJ / . /

2. Trigrams that match the corpus are extracted. This is what is known as statistical processing. Any trigrams that contain the KEY tag are multiplied – this is a rule added to tweak performance.

“My old phone”
PP$ JJ NN
Probability: **137.0**

“phone was terrible”
NN BEDZ JJ
Probability: **212.0**

“is the iphone”
BEZ DT KEY
Probability: **50.0**

“iphone is amazing”
KEY BEZ JJ
Probability: **450.0**

“iphone is amazing”
KEY BEZ JJ
Probability: 450.0

3. The most likely trigram is searched for any words that contain polarity (as defined in the word list).

“amazing”
Wordlist = Positive Word

4. The correct polarity is found – Positive.

Tweet is Positive

As mentioned earlier, SMME utilises statistics and rules to determine polarity. Because SMME keeps track of the syntactical data of each tweet being processed it can easily have rules added to improve the accuracy.

In *figure 2* there is already one rule integrated and takes place at step three. When a trigram is found which contains the lexical semantic KEY, the probability of using this trigram is multiplied by a constant. This simple rule is easy to implement and gives a bonus to any opinionated words very close the to the keyword.

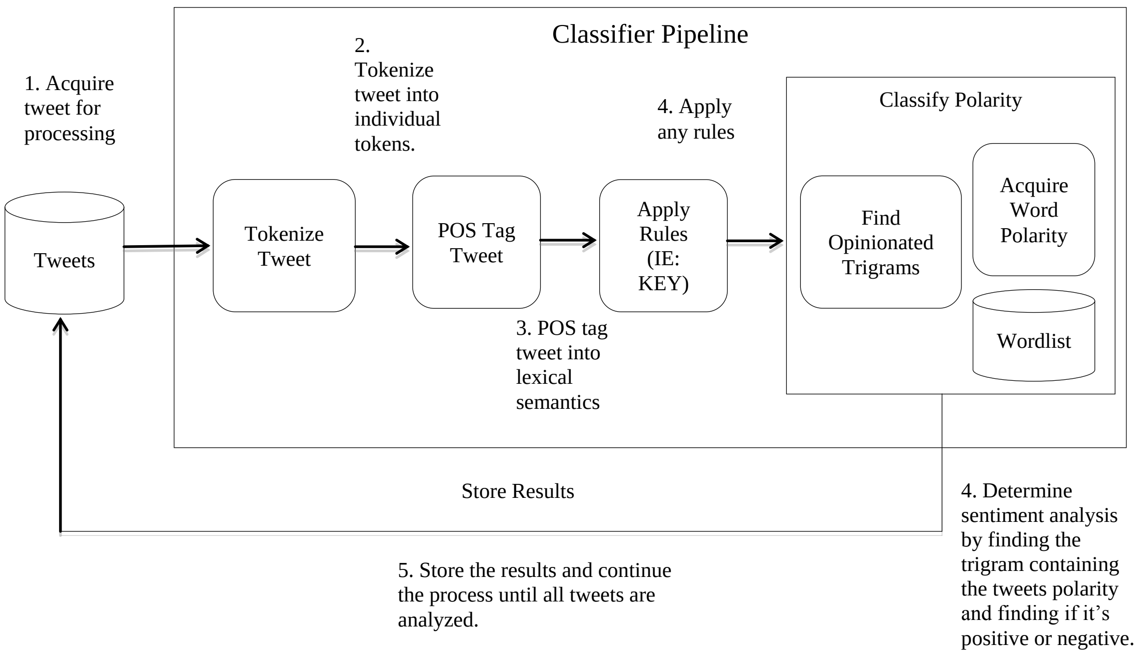
The fact the classifier acknowledges the syntactic structure of a tweet and is part rule based, the integration of new rules can potentially improve classifier results dramatically.

Example rule additions:

- Distance between the trigram and the term being searched for. The closer the distance to the term the more likely it is to be the trigram containing polarity.
- Only use the sentence in a tweet that contains the term.
  - *Example: I like my cat. I really hate the weather.* *Keyword: cat.*
- Specific rules on conjunctions. IE: Only search the latter part of a sentence when the word 'but' is used.
- If the trigram contains 'not' (*RB*) prior to the opinionated word in the trigram, reverse the polarity. This solves the issue of double negatives.

Diagram of SMME classifier pipeline:
Figure 3

The classifier is designed to process tweets in a 'pipeline'. A tweet is inserted, pushed along a sequence of stages and returned with an assigned sentiment. The benefit of using a pipeline allows future expansion, easy debugging and speed increases (if needed) by using threads to queue each step.



Classifier Pipeline
1. Acquire tweet for processing
2. Tokenize tweet into individual tokens.
4. Apply any rules
Classify Polarity
Tweets
Tokenize Tweet
POS Tag Tweet
Apply Rules (IE: KEY)
Find Opinionated Trigrams
Acquire Word Polarity
Wordlist
3. POS tag tweet into lexical semantics
Store Results
4. Determine sentiment analysis by finding the trigram containing the tweets polarity and finding if it's positive or negative.
5. Store the results and continue the process until all tweets are analyzed.

5.3 Automated Corpus Generation

The corpus provides a **very** important role for any statistical classifier. It is the foundation of knowledge the classifier uses to determine any assessments. The data inside a corpus can range from a magnitude of different alterations dependent on the cause of the classifier.

In any form of statistical analysis of text, the corpus must be relevant to the type of text that is being evaluated.

Example:

- Tweets for Tweet SA
- Movie Reviews for Movie Reviews SA
- Spam Examples for Spam Detection
- Correct Spellings for Spelling Error Detection

This is very important for two reasons.

First, the classifier is trained on this data and as such it tells the classifier what it is looking for. It is impossible to train a spam detection engine on data from movie reviews. The data inside a movie review does not have any correlation to what spam is or isn't. To determine what spam is, the classifier must have knowledge of what is and isn't spam and as a result, the classifier must be trained on the relevant data.

Second is domain dependent syntax. It may seem possible to train a sentiment analysis engine on any text that includes positive, negative and neutral defined data; this would offer worse accuracy. Each cultural or context domain uses different syntax in the English language. The text style used inside a text message is different to the text used inside an email. The same can be applied to tweets which also include a new form of syntax, one that which incorporates hash tags, links, malformed grammar, emoticons and some slang. As a result, the data used for training a Twitter sentiment analysis engine must be made of tweets.

As previously mentioned a negative point for the Naïve Bayes or other statistical solutions require a manually defined corpus. In some studies it is possible to semi-automate the process of building a corpus from human defined data such as movie reviews (Pang, Lee, & Vaithyanathan, 2002); however, in the majority of cases a corpus must be created by a human. This is time consuming considering a good corpus is in the range of thousands if not millions of entries. A feature of SMME is its ability to create a corpus automatically with no human interaction. In essence, the script could be left to run indefinitely to build a large and efficient corpus.

There are two 'phases' of autonomously building a corpus for SMME's classifier. Tweets that are deemed subjective need to be extracted from those that are not. These tweets are then processed for any trigrams that contain polarity, which are then used to build up the actual corpus.

5.3.1 Gathering Subjective Tweets

The automated corpus generator for SMME ties in very closely with how the classifier works and is built completely different to Naïve Bayes corpus requirements. Assuming Naïve Bayes is used for twitter SA, it would require a list of bigrams (or variation) that appear in positive, negative and neutral tweets. SMME classifier however just requires a list of trigrams that are present in subjective tweets – essentially a tweet containing any opinion.

This is a defining feature as it enables a corpus to be automated with good end results. It is impossible to make a generated corpus containing positive, negative and neutral findings for Naïve Bayes because if it could there would be no need for a classifier or Naïve Bayes in the first place. The work of Dr. Turney influenced a method on how to extract data that has a very high chance of being opinionated. The method involves extracting tweets that contain trigrams known to commonly occur in subjective texts.

The SMME corpus generator processes *n* amount of tweets and keeps any that contain a trigram in the list documented by Dr. Turney, including any extra personal additions. At this point it is possible to assume the majority of tweets that get through do contain polarity of some form.

Trigrams containing polarity
Table 3

Trigrams proposed by Dr. Turney in the 2002 proceedings.

First	Second	Third
JJ	NN|NNS	ANY
RB|RBR|RBS	JJ	NOT NN|NNS
JJ	JJ	NOT NN|NNS
NN|NNS	JJ	NOT NN|NNS
RB|RBR|RBS	VB|VBD|VBN|VBG	ANY

(Turney, 2002)

5.3.2 Spam Filtering

Any spam that is present inside the corpus can lead to an undesirable reduce in classifier accuracy. This is due to consistent (generally biased positive) structure of tweets that pass through. If the corpus is built up heavily of spam, it is possible lots of spam will be deemed positive when performing sentiment analysis as the classifier will be trained using a spam heavy corpus. Filtering is a simple (yet crude) method of removing any tweets that consist of certain regex. An example of this is removing any tweets that contain website addresses.

The list of filtered spam regex:
Table 4

Regex	Description
"RT"	*Re-Tweets are duplicated Tweets deemed important by the public.*
"(http|www|/.com|/.co/.uk)"	*Any link that contains a web address usually is spam.*
"#"	*OPTIONAL: Removal of anything using tags. Often this includes directed spam.*
"@"	*OPTIONAL: Sometimes targeted (replied) tweets don't contain any useful data for corpus generation.*

Diagram of SMME Corpus Processing example:
Figure 3

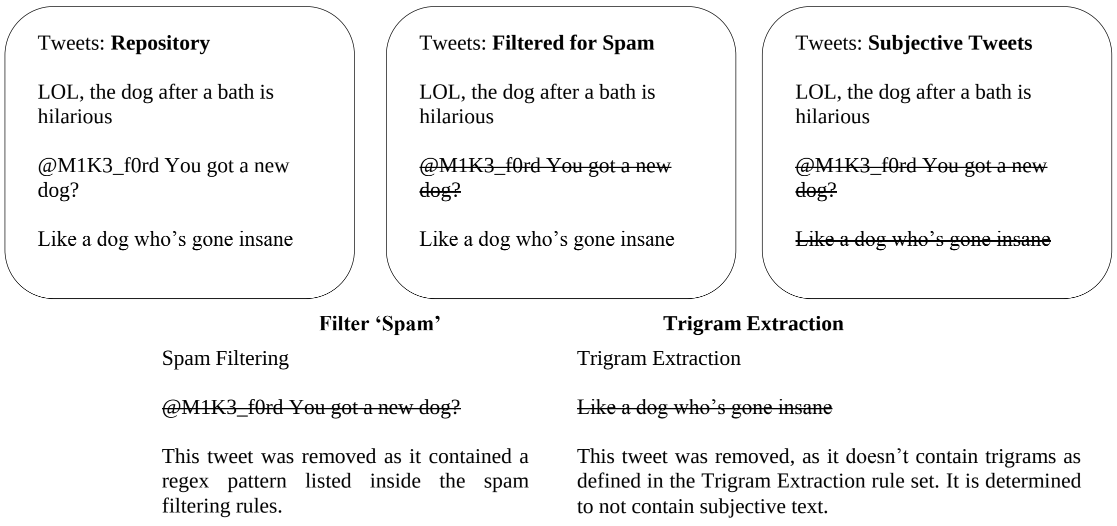
This diagram clearly indicates how SMME automatically extract tweets it deems as subjective. Tweets are filtered for spam and then the trigram extraction method is applied.

Example Trigrams used for Extraction

First	**Second**	**Third**
JJ	NN | NNS	ANYTHING
NN | NNS	JJ	! NN | NNS
VBZ	JJ	ANYTHING

Example Rules used for Spam Filtering

@
RT



Tweets: **Repository**

LOL, the dog after a bath is hilarious

@M1K3_f0rd You got a new dog?

Like a dog who's gone insane

Tweets: **Filtered for Spam**

LOL, the dog after a bath is hilarious

~~@M1K3_f0rd You got a new dog?~~

Like a dog who's gone insane

Tweets: **Subjective Tweets**

LOL, the dog after a bath is hilarious

~~@M1K3_f0rd You got a new dog?~~

~~Like a dog who's gone insane~~

Filter 'Spam'

Trigram Extraction

Spam Filtering

~~@M1K3_f0rd You got a new dog?~~

This tweet was removed as it contained a regex pattern listed inside the spam filtering rules.

Trigram Extraction

~~Like a dog who's gone insane~~

This tweet was removed, as it doesn't contain trigrams as defined in the Trigram Extraction rule set. It is determined to not contain subjective text.

5.3.3 Corpus Construction

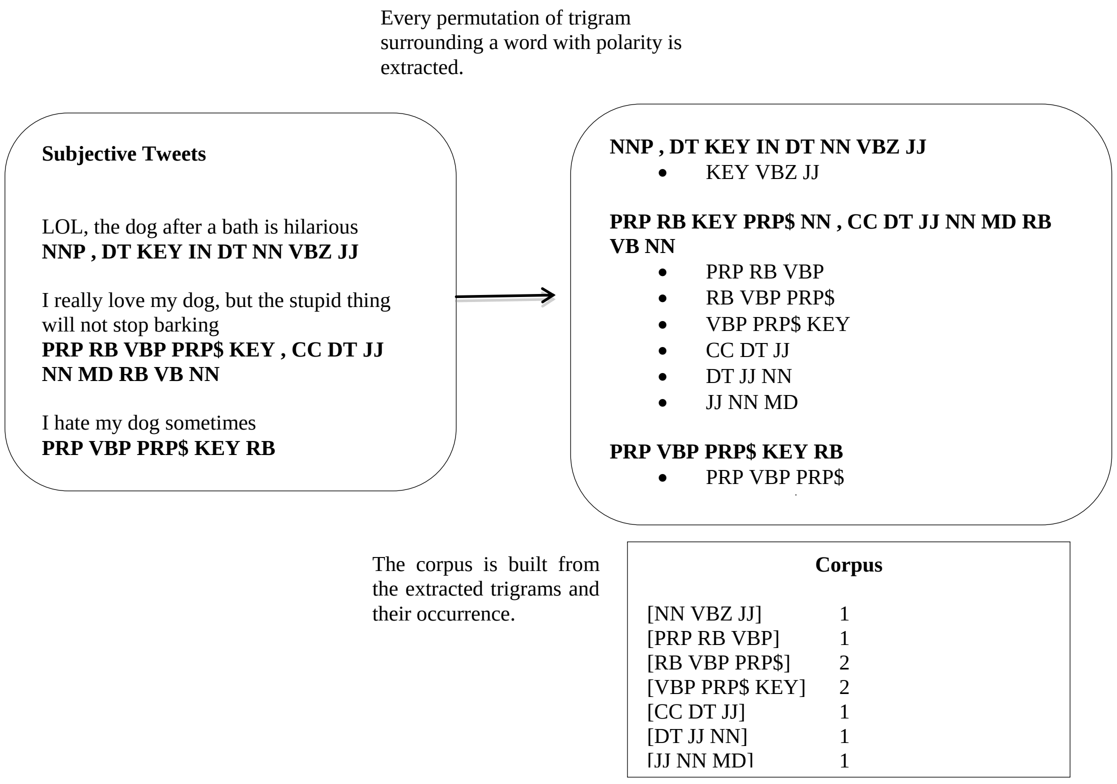
The classifier is dependent on a specialized format of corpus. It requires a corpus consisting of trigrams and their occurrence rate in the following format:

[JJ, NN, NN] 22

This informs the classifier the trigram [JJ NN NN] contains polarity for a sentence and takes a higher precedence than a different trigram with an occurrence lower than 22. To convert a long list of tweets into this format is a relatively simple processes.

Diagram of SMME Corpus Building example:
Figure 4

Each tweet that has been deemed subjective by the previous method is searched for every permutation of trigram that contains a word with polarity. The trigram is added to a list including the amount of times it has occurred. A finished SMME corpus will be the end result.



Every permutation of trigram surrounding a word with polarity is extracted.

Subjective Tweets

LOL, the dog after a bath is hilarious
NNP , DT KEY IN DT NN VBZ JJ

I really love my dog, but the stupid thing will not stop barking
PRP RB VBP PRP$ KEY , CC DT JJ NN MD RB VB NN

I hate my dog sometimes
PRP VBP PRP$ KEY RB

NNP , DT KEY IN DT NN VBZ JJ
- KEY VBZ JJ

PRP RB KEY PRP$ NN , CC DT JJ NN MD RB VB NN
- PRP RB VBP
- RB VBP PRP$
- VBP PRP$ KEY
- CC DT JJ
- DT JJ NN
- JJ NN MD

PRP VBP PRP$ KEY RB
- PRP VBP PRP$

The corpus is built from the extracted trigrams and their occurrence.

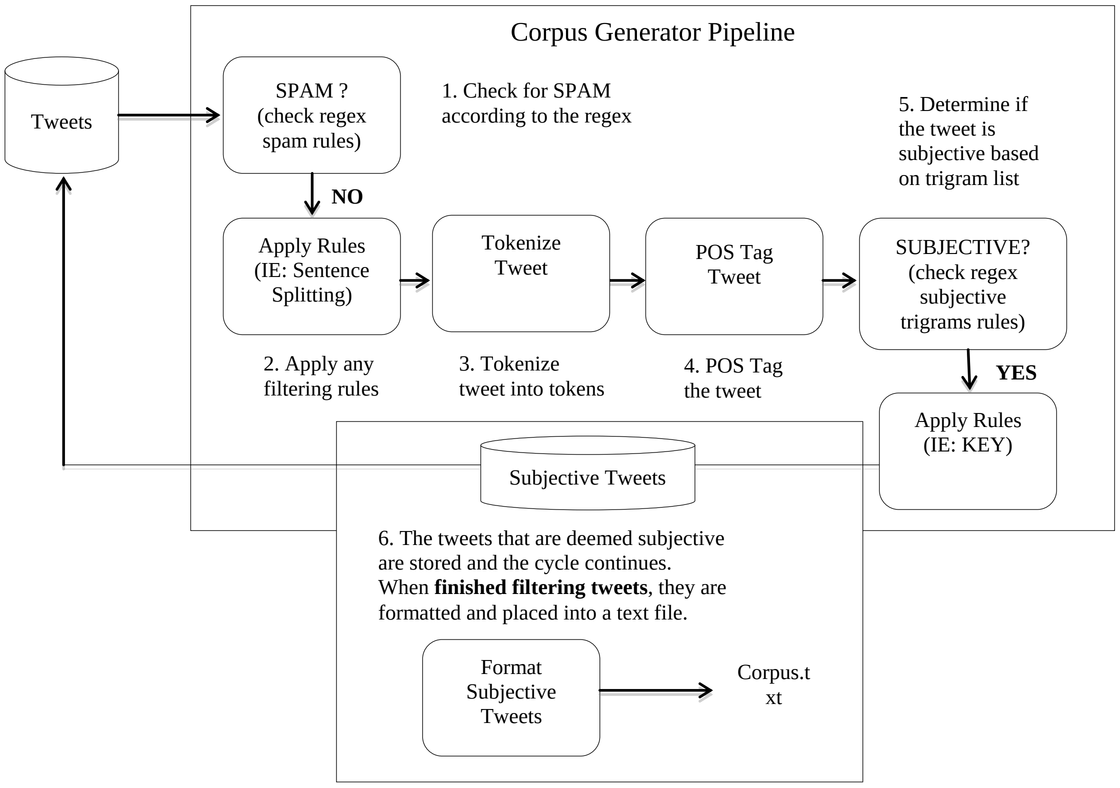
Corpus

[NN VBZ JJ]	1
[PRP RB VBP]	1
[RB VBP PRP$]	2
[VBP PRP$ KEY]	2
[CC DT JJ]	1
[DT JJ NN]	1
[JJ NN MD]	1

A final addition to any Tweet is the 'term' used is replaced with the tag KEY. There are two reasons why this is useful and important. The first means that any term that may contain an opinionated keyword (IE: The Great War) will skew the corpus; the word 'Great' would be seen as an adjective rather than part of the term. It also means in future additions a rule could be added that requires the location of the KEY within the Tweet and makes it easily accessible.

<u>Diagram of SMME corpus generator pipeline:</u>
Figure 4

The corpus generator is designed as a pipeline for the same reasons as the classifier. As one can see there are two distinct parts, the filtering of subjective tweets and the construction of the corpus text file using those filtered tweets.


Corpus Generator Pipeline
Tweets
SPAM ? (check regex spam rules)
1. Check for SPAM according to the regex
5. Determine if the tweet is subjective based on trigram list
NO
Apply Rules (IE: Sentence Splitting)
Tokenize Tweet
POS Tag Tweet
SUBJECTIVE? (check regex subjective trigrams rules)
2. Apply any filtering rules
3. Tokenize tweet into tokens
4. POS Tag the tweet
YES
Apply Rules (IE: KEY)
Subjective Tweets
6. The tweets that are deemed subjective are stored and the cycle continues. When **finished filtering tweets**, they are formatted and placed into a text file.
Format Subjective Tweets
Corpus.txt

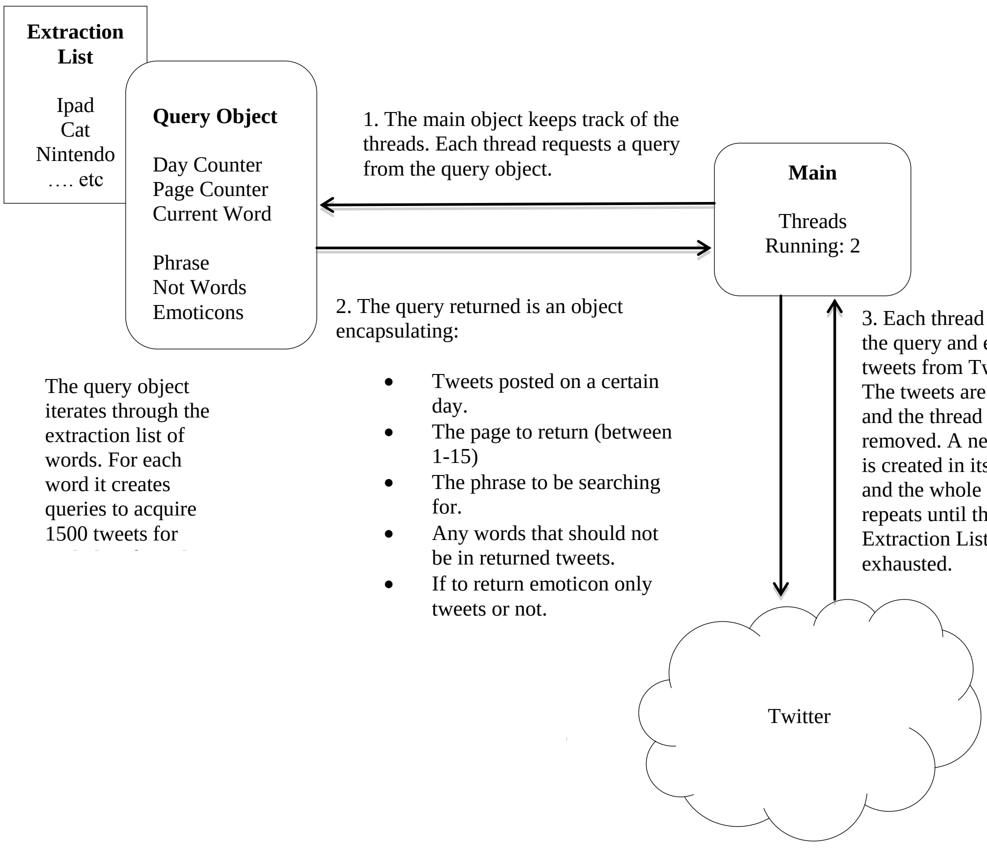
5.4 Tweet Extractor

Extracting tweets is an important section of SMME. It is required for obtaining tweets to build a corpus from and tweets to perform sentiment analysis on. The extractor is designed as an independent system to improve modularity between both requirements.

The Twitter extractor uses the Twitter Search API. It allows SMME to search back up to five days and acquire 1,500 tweets for each day. The API uses a URL GET based middleware and responds in JSON objects. The returned JSON object is iterated and placed into individual Tweet objects. These can be written to a database or into any other temp storage (arrays).

To achieve efficiency the extractor utilizes multiple threads to concurrently call the Twitter API for tweets. The speed depends on how many tweets are required, how many days back and how many threads are running. It has been documented speeds of around four seconds to return 7,500 tweets over four days using 30 threads.

Diagram of SMME Tweet Extractor example:
Figure 5


Extraction List
Ipad
Cat
Nintendo
…. etc
Query Object
Day Counter
Page Counter
Current Word
Phrase
Not Words
Emoticons
1. The main object keeps track of the threads. Each thread requests a query from the query object.
Main
Threads Running: 2
2. The query returned is an object encapsulating:
• Tweets posted on a certain day.
• The page to return (between 1-15)
• The phrase to be searching for.
• Any words that should not be in returned tweets.
• If to return emoticon only tweets or not.
3. Each thread executes the query and extracts tweets from Twitter. The tweets are stored and the thread is removed. A new thread is created in its place and the whole cycle repeats until the Extraction List is exhausted.
The query object iterates through the extraction list of words. For each word it creates queries to acquire 1500 tweets for
Twitter

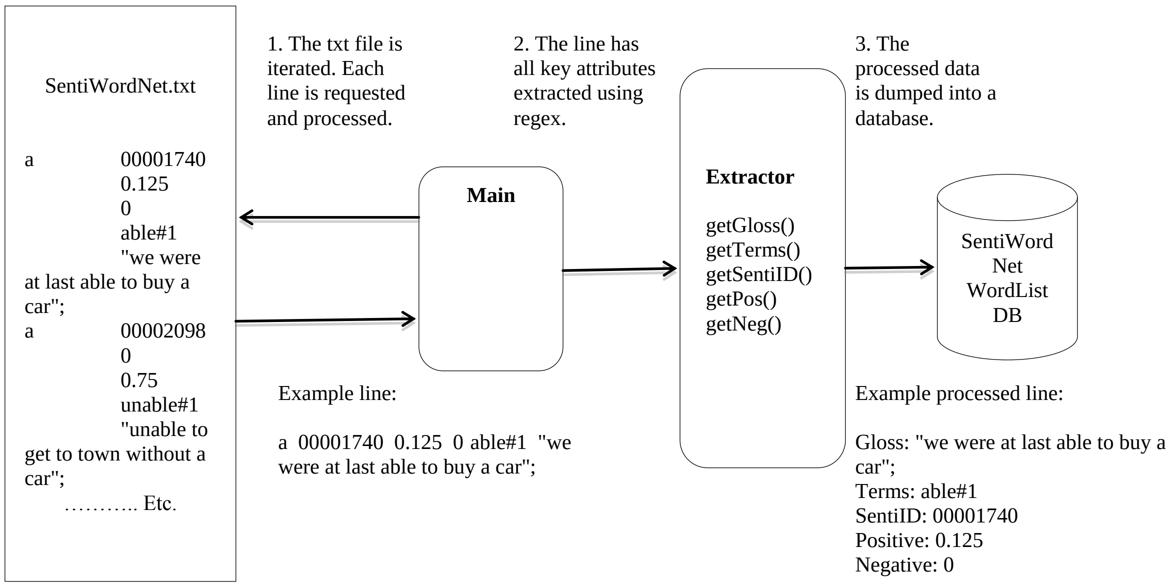
5.5 Wordlist

SMME works differently to other classifiers and requires the use of a wordlist to determine what words contain polarity. When the classifier determines a trigram contains the sentiment for the tweet, the trigram needs to be assessed for sentiment. To do so, each word from the trigram is checked for polarity. Considering computers have no understanding of human text a database full of words and their polarity (negative and positive) is required, this is the wordlist.

The information inside this wordlist can be gathered from any source. SMME primarily focuses on the use of SentiWordNet for its reputability of quality and sheer quantity of information it contains. The data provided by SentiWordNet is presented within an irregular formatted text file. To populate a database with this data a Java application was designed.

Diagram of SMME SentiWordNet DB Population:
Figure 6

Population of the database using SentiWordNet data is a combination of text iteration and regex segregation. Each string line of text is extracted and has a sequence of regex rules applied to it. Relevant data is stripped using the regex and pushed into the database.

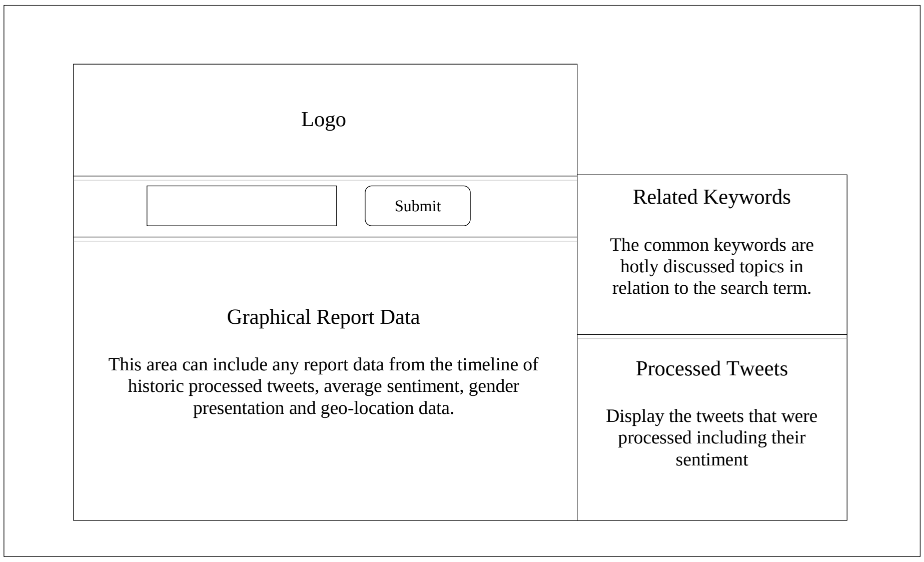

SentiWordNet.txt
a 00001740 0.125 0 able#1 "we were at last able to buy a car";
a 00002098 0 0.75 unable#1 "unable to get to town without a car";
……….. Etc.
1. The txt file is iterated. Each line is requested and processed.
2. The line has all key attributes extracted using regex.
3. The processed data is dumped into a database.
Main
Extractor
getGloss()
getTerms()
getSentiID()
getPos()
getNeg()
SentiWord Net WordList DB
Example line:
a 00001740 0.125 0 able#1 "we were at last able to buy a car";
Example processed line:
Gloss: "we were at last able to buy a car";
Terms: able#1
SentiID: 00001740
Positive: 0.125
Negative: 0

5.6 Website

The website for SMME is a graphical user interface for the user allowing them to enter a term into SMME and retrieve a report. The report is generated live by the website using data processed by the SMME sentiment analysis engine. The website is built using XHTML, CSS, JSP, JavaScript, JSON and Java Servlet technologies all hosted on a local Tomcat server.

Diagram of SMME Website
Figure 7

The design of the website clearly displays the requirements of SMME with enough room for future additions.


Logo
Submit
Related Keywords
The common keywords are hotly discussed topics in relation to the search term.
Graphical Report Data
This area can include any report data from the timeline of historic processed tweets, average sentiment, gender presentation and geo-location data.
Processed Tweets
Display the tweets that were processed including their sentiment

5.6.1 Servlet

The servlet is an important role of the website by executing concurrent requests by different users. Each time a user searches a request it is sent to the servlet, sending the search term via POST. The servlet creates a Tweet Extractor object to retrieve tweets for the given search term and forwards the results on to a newly created Classifier object. Once the processing is complete, the servlet injects attributes into the users session object and redirects them to a new URL. This report dynamically generates the report using JavaScript and JSP. It is possible to view an overview of this process in *figure 1*, located in the Design chapter’s introduction.

6. Implementation

The implementation chapter describes the errors occurred from following the implicit design schematics, but also how the design has evolved since the initial report through development changes, problems and impracticalities.

The designs themselves have evolved through the implementation process due to impossibilities and inefficiencies discovered on route of development. The biggest issues implementation presented were the consistent failure of API's not performing to their specifications and as a result, extensive resources into solutions or workarounds were wasted. Furthermore, time resources rapidly depleted in implementation due to the lack of knowledge in certain technical areas such as web development. In this chapter, the main issues are revealed and discussion of their solutions. The evaluation (results) of completed code also had a major impact on design, but will be covered in the next chapter.

6.1 Third Party Library Integration

SMME utilises multiple external libraries to fulfill design requirements.

Library	Summary	Description
OpenNLP (Ingersoll, 2010)	Tokenizer	Splits text into individual tokens.
QTagger (Mason, 2010)	Part of Speech Tagger	Attaches the lexical semantic category to a token (IE: Noun to the token "Car")
Twitter4J (Yusuke, 2007)	Twitter Integration Library	Powerful API used to search and extract tweets from Twitter.
Gson (Leitch, 2008)	Json	Simplifies the use of Javascript Object Notation.
JUnit (Beck, 2009)	Java Testing Platform	Enables the creation of test cases to evaluate system reliability.
PostGres Driver (PostGres , 2004)	Database	Enables interaction with PostGres databases from Java code.
Tomcat (Apache, 1999)	Server	The Java based website is emulated on a local host Tomcat server.
SentiWordNet (Baccianella, Esuli, & Sebastiani, 2010)	Sentiment Wordlist	A list of words containing sentiment.
SentiStrength (Thelwall, Buckley, Paltoglou, & Cai, 2010)	Sentiment Wordlist	A list of words containing sentiment.

6.2 Classifier

Class Diagram can be viewed in Appendix A.
Sequence Diagram can be viewed in Appendix A.

Implementation of the classifier is built in Java. The module is currently built specifically for the use of SMME's servlet but can easily undergo modifications of its constructor to allow extra parameters or return new values. The classifier requires a corpus to work and an arraylist of tweets, after processing it allows a developer to extract the processed tweets in a JSON object and multiple values for report rendering (IE: quantity of positive tweets on Monday).

6.2.1 Implementation Problems

Remarkably during implementation of the classifier no problems emerged, this was most likely due to extensive research and 'dry' runs conducted to predict plausibility. It is worth noting however, the design of the classifier evolved slightly from the interim report to its current form due to implementation efficiency and end result feasibility.

- Use of MLE average in training data
    - The original plan was to use MLE (Maximum Likelihood Estimate) to establish the importance of each trigram inside the training data (corpus). It is quicker, less error prone and no difference in results to simply use the quantity of trigram occurrence.
- Slight changes to training data and classifier integration
    - A prominent idea in the logbook suggests keeping track of the whole syntactical data from tweets that make up the training data. The theory was, if a trigram inside the corpus could not be matched to the tweet being classified then bigrams could be searched for. This method allows the option to start from any n-gram and keep depleting down to a bigram until a match was found. This may have improved results but because of lack of time resource and unsure on efficiency gain it was dropped.

6.2.2 Testing

The classifier uses JUnit for testing because it is a non-trivial component of the SMME architecture and it handles live data. In order to test the classifier is sanitized against most (all) permutations of data handling multiple tests were conducted in JUnit.

All exceptions are handled using custom exceptions and 'bubble' up to the main object class so the servlet can easily distinguish where and if an error is thrown.

6.3 Automated Corpus Generation

Class Diagram can be viewed in Appendix B.
Sequence Diagram can be viewed in Appendix B.

The corpus generation design is implemented in Java and is a main executable. It requires a PostGres database of tweets to build the corpus from. The code only needs to be run once to create a corpus, the classifier can then use this corpus indefinably. The finished corpus is a text file inside the codes directory.

6.3.1 Implementation Problems

The accuracy of SMME's classifier greatly depends on the corpus it uses. The classifier is trained on the corpus and as a result, if the corpus is wrong the classifier will be to. The following simple changes to the corpus generation design improved the classifiers accuracy significantly.

6.3.1.1 Key Term

The original designs of the classifier and corpus generation introduced a horrific bug. If a term being searched for contained polarity (IE: The Great Escape), the report would use the polarities found inside the term; in this case, all tweets would be positive due to 'Great'. To overcome this the Key Term rule process was integrated into the corpus generator and classifier.

When collecting subjective tweets to build a corpus from, just after POS tagging the searched term for that tweet must have its tag replaced to KEY.

Example:
Tweet: I hate the film love actually
Before: PRP VBP DT NN NN RB   After: PRP VBP DT NN KEY

This has the following purposes:

- The lexical semantic meaning for the words (love actually) are stripped away and replaced with KEY.
    - A corpus being built on tweets for the film 'love actually' can be used for any Twitter sentiment analysis as KEY is universal.
    - Removing the lexical semantics stops false positives when finding subjective trigrams.

6.3.3.2 Sentence Splitting (Context Ambiguity)

Ambiguity is the most difficult problem for the all NLE systems to overcome and as of yet, all SA methods can’t process it; the automated corpus generation process for SMME can’t either. The corpus generation code works on a ‘blind’ basis; it doesn’t know the context of the words and simply produces results from a sequence of rules. Because the code works in this manner, lots of subjective tweets are missed and an equal share of ambiguous irrelevant tweets get through. The lack of context evaluation is the main reason why the accuracy of the corpus generator cannot achieve a very high accuracy rate but it is also something currently impossible to achieve.

Example:
Keyword: iPhone
Tweet: Twitter for the iPhone is rubbish

A human can easily establish that the derogatory term ‘rubbish is in fact referring to the app ‘Twitter’ for the iPhone. The computer has no concept of words, their relation or even their context. Instead, this is treated as a subjective tweet for the term ‘iPhone’ because it caters to the rules:

- It contains a trigram of subjective statement “*iPhone is rubbish*”.
- It has the term ‘iPhone’ in the same sentence as the trigram
- It doesn’t contain any regex associated with spam.

Of course, this judgment is wrong as the term ‘iPhone’ is not the context of this tweet.

The introduction of sentence splitting improved the context ambiguity and overall accuracy by a significant factor. Concurrent observation into improvement deficiencies showed lots of tweets meeting the criterion that defines a subjective tweet (containing subjective trigrams and the keyword) but as described above in ‘Ambiguity’ does not actually qualify when assessed by a human.

Sentence splitting helped reduce context ambiguity in certain situations. Instead of looking at the whole tweet, the corpus generator just looks at the sentences of a tweet that contain the keyword. Doing this improved the accuracy of the generated corpus.

Example:
Keyword: dog
Tweet: I have a pet dog. I hate it when it rains because I have to walk it.

The corpus generator would extract this tweet as being opinionated because it includes a trigram holding polarity and the term. A human can easily see that the negativity is actually aimed at the rain, rather than the dog. The sentence splitting technique looks at the sentence with the term in: ‘*I have a pet dog’*. Doing so the system no longer believes this tweet is subjective towards the term dog and as a result, ignores it.

The following experiment shows how sentence splitting increases the accuracy of determining what tweets are classed as subjective (Qualified Tweets).

Each test uses the following filtering parameters:

Tagger: qTag. Tokenizer: openNLP

Spam:

- @
- RT
- Web Links
- #

Trigram Rules:

First	Second	Third
JJ	NN|NNS	ANY
RB|RBR|RBS	JJ	NOT NN|NNS
JJ	JJ	NOT NN|NNS
NN|NNS	JJ	NOT NN|NNS
RB|RBR|RBS	VB|VBD|VBN|VBG	ANY

Experiment 1 – Improvement of Sentence Splitting on Corpus Generation

100 Tweets are processed with the automated corpus generator. The percentage in the results below is the amount of tweets assessed correctly within that category.

Results **WITHOUT** sentence splitting:

	Qualified Tweets	Failed Tweets	Spam Tweets
Tweets with Sentiment	**57.14%**	33.33%	12%

Results **WITH** sentence splitting:

	Qualified Tweets	Failed Tweets	Spam Tweets
Tweets with Sentiment	**70%**	63.15%	12%

As one can see the amount of subjective tweets is increased by 13 percent for qualified tweets, but it is also increases dramatically for failed tweets. This is to be expected, as true context ambiguity cannot be solved with a simple rule. In some cases it will work and in others it will have a negative effect. This is one reason why fully rule based classifiers will not perform well due to the amount of different permutations text can come in, all of which would need to be covered. The amount of subjective tweets that are being discarded under 'failed tweets' does not matter; the quantity of tweets in twitter means data is expendable. Because this technique is used for corpus generation only, improving the qualifying tweets accuracy is far more important than reducing the amount of tweets which fail to pass.

6.3.2 Testing

Testing of the reliability of the code is not required. It is run once by the developer to acquire a corpus for the classifier and is not designed for a public user interaction. It is assumed if programmers can execute the corpus generator, they also will have experience to fix any bugs. To assist in debugging all exceptions are custom made to help point towards errors, however out of all the runs the only errors that occur are due to false database connection information.

6.4 Tweet Extractor

Class Diagram can be viewed in Appendix C.
Sequence Diagram can be viewed in Appendix C.

The tweet extractor for SMME is a very important role. It is a java component that extracts tweets from twitter using a list of words. This list of words can be any size; the extractor iterates the list, searches for that term and populates a database with any results found. It is created in Java and can be used by either submitting a String term via the constructor or hooked up to a text file of terms to iterate. The quantity of threads, days to retrieve and time intervals can all be managed using the objects runtime variables interface.

6.4.1 Implementation Problems

6.4.1.1 Tweet Extraction

SMME requires a large amount of tweets from Twitter in two loads, one large intermittent call every time the user searches and one very large continuous search for collecting corpus data. To achieve this SMME needs to use an API for collecting tweets. Exhaustive research into the field shows there is a limited selection available. This became the biggest burden throughout the whole project as the main Twitter API was relied on which as soon explained contains explicit problems.

The available API's to collect tweets come in two types; they either rely on the Twitter API and expand around it or (mostly) collect their own database of tweets over a long period of time and supply their own API for accessing them. There are lots of websites that have their own databases of tweets but don't have a developer API to use their system.

Different Websites Using Twitter Data
Table 5

Website	Cost	SMME Applicable	Rate Limit	Accessible API
Searchtastic	Free	Yes	-	No
SnapBird	Free	Yes	-	No
TwimeMachine	Free	Yes	-	No
Topsy	**Free**	**Yes**	**100 per hour**	**Yes**
The Archivist	Free	No	-	Yes
BackTweets	$100 per month	Yes	-	Yes
Research.Ly	$ 9 per month	Yes	-	Yes
Twitter Search API	**Free**	**Yes**	**Unkown**	**Yes**

(Larson, 2010)

Even though there is a large amount of websites providing services of *historic* tweets (tweets from days passed) and containing all the features SMME requires, they do not provide an API to use their data/systems or if they do, it costs money. Twitter Search and Topsy became the primary API's.

API's with Precedence
Table 6

To conclude the best API, further research and experiments into them commenced. These include retention, location, date and speed. The speed data was achieved by retrieving 50 tweets using their system in a controlled environment and recorded in milliseconds.

API	Retention	Location Data	Date Data	Speed
Topsy	100	No – Pragmatic	No – Pragmatic	1588 ms
Twitter Search	7	Yes	Yes	800 ms

* No – Pragmatic: It is possible to retrieve this data, but instead of supplied by the API requires extensive programming. IE: To retrieve location, need to acquire user ID from the tweet and search for user's profile on Twitter.

The extensive requirement of more programming and slower speed meant Twitter Search the most predominant API to use. This was further amplified once taking into account their rate limits.

6.4.1.2 Rate Limitation

Rate limiting is the limitations on how many calls a user can make from an IP address per timeframe, IE: 200 calls per hour. Due to the quantity of tweets SMME requires the following problems emerged during implementation:

- If too many users search per hour on SMME using a rate limited API, the IP gets blacklisted.
- Acquiring 100,000 tweets for a corpus would take ten hours when using an API that gets limited after 10,000 tweets per hour.

Both API's supplied by Topsy and Twitter Search contained rate limits. Considering the Topy API is very limited (100 calls per hour * 100 tweets per call = 1,000 tweets per hour) the Twitter Search is the most applicable.

To attempt to obtain what the Twitter Search API's rate limit is (isn't documented) extensive experiments into test cases were made.

EXPERIMENT 1 – Assessment of Rate Limiting for Twitter Search API

Each test was a search for the keyword 'ipad'.
A 'call' returns 50 tweets. A new thread was run for each 'call' to achieve maximum speed. The script would not stop until a "rate-limit exceeded exception" was thrown.

Time Executed	Successful Calls	Time Blacklisted For
13.45	1998	06:00
13.51	2025	04:00
13:55	2028	02:00
13:57	1724	25:00
14:22	1978	01:00
14:25	1957	02:00
14:28	1972	01:00
14:31	1637	21:00

The times between blacklisting and calls allowed are not consistent. It seems from this analysis the rate limit is reset every 20 – 25 minutes.

1998 + 2025 + 2028 + 1724 = 7,775
1978 + 1957 + 1972 + 1637 = 7,544

(7, 775 + 7544) / 2 = **7659.5**
(25 + 21) / 2 = **23**

The average calls per **23** minutes are **7660**.

Initially this may seem like a very high calls:time ratio however the API has an additional feature not expressed in this experiment; the Anti-DDOS system.

EXPERIMENT 2 – Assessment of Rate Limiting and Anti-DDOS for Twitter Search API

The average calls per **23** minutes are **7660**.

x = 7660 / 60
x is how many calls a user should be able to make per second in one minute before being blacklisted.
Incorrect: The Anti-DDOS System kicks in after eleven seconds (1397 calls).

x = 7660 / 23
y = x / 60
y is how many calls a user should be able to make per second before being blacklisted.
Incorrect: The Anti-DDOS System kicks in after two minutes (600 calls).

A test was made on calling the API once every second. This will only result in 1,380 calls in 23 minutes and should not blacklist.
Incorrect: The Anti-DDOS System kicks in after three minutes (180 calls).

The tests show conclusive evidence the Twitter API not only has some form of **x** tweets in **y** timeframe but also an Anti-DDOS system. If the Twitter Search API records a user accessing the API in a consistent fashion (as shown, once a second) it treats it as a DDOS attempt and blocks all connections from that IP for a period of time. This presents the following problems:

- Still no clarification what the Twitter API rate limit is
- Problem with user searches on SMME, too many or too consistent in a small timeframe will result in a black list.
- Consistent running of a script to obtain tweets for the corpus will result in a black list.

In an attempt to fool the Anti-DDOS system, a new experiment was conducted.

EXPERIMENT 3 – Assessment of Beating the Anti-DDOS System for the Twitter Search API

Each test consisted of *n* threads calling the Twitter Search API. To try and fool the Anti-DDOS system, each time a thread finishes it waits for a random time (1-10 seconds) before calling again. This is an attempt at mimicking natural data calls.

Test #	Threads	Calls	Time Running	Time Lasted	Rate Limited
1	1	615	30:00	30:00	No
2	2	1086	30:00	30:00	No
3	4	2345	30:00	30:00	No
4	8	2334	30:00	17:00	Yes
5	6	2176	30:00	21:00	Yes
6	5	2131	30:00	24:00	Yes
7	4	4560	60:00	58:00	Yes

The more threads that are added increase the chance of detection due to the element of ‘natural’ individual calls has a higher chance of becoming overlapping calls that the Anti-DDOS system picks up. It is impossible to predict the rate limit and the extent of the Anti-DDOS system of the Twitter Search API without a large study and an abundance of test cases; however, the tests conducted did provide a solution to two of the requirements of SMME.

To conclude, the Twitter Search API can be called using two threads per hour on a 10 second random interval timer without being subjected to rate limiting. This enables the twitter extractor to run consistently to obtain tweets for the corpus generator.

2 Threads = 1086 calls per 30 minutes
2 Threads = 2172 calls per 60 minutes
1 call = 100 tweets
*2172 calls * 100 = 217,200 tweets per hour*

In terms of solving the issue with multiple users using SMME in quick succession (each search using SMME requires up to 7,500 tweets – or 75 calls) it is impossible to work out how many threads can be run to retrieve the tweets required in the shortest time. This research however did show that the approach of random interval threads does reduce the odds of being rate limited.

6.4.1.3 Twitter4J Legacy Code Compilation

To access Twitter a third party library Twitter4J is used. The library provides easy access to tweets via the Twitter Stream, Search or Rest API's. A fundamental error occurred during testing SMME using 'emotional' tweets and searches containing one or more words ("a phrase, IE: 'Sega Megadrive'). The API does not support this by default so the legacy code of Twitter4J required amending and re-compilation.

The changes made:

- Enable search of tweets using the 'emoticon' GET attribute
- Enable search of tweets using the 'not' GET attribute
- Enable search of tweets using the 'phrase' GET attribute

6.4.2 Testing

JUnit test cases are used to test the Twitter Extractor for errors ranging from null data, rate limitations and quantity of threads. Due to the importance of this component, all exceptions are 'bubbled' up to the servlet layer so a relevant error page can be displayed to the user.

6.5 Word List

The original design schematics for the word list changed throughout the implementation stage. The word list is used for determining the polarity of a word; therefore for a wordlist to be acceptable it requires a list of words and their polarity (if positive or negative). As noted in the design chapter SentiWordNet is the wordlist of choice; however, during implementation and evaluation it was deemed unacceptable due to its quality and compatibility with SMME. An alternative wordlist is used for both comparative evaluation and increase accuracy (Thelwall, Buckley, Paltoglou, & Cai, 2010). This alternative wordlist called SentiStrength resulted in becoming the primary source of polarity words.

The ER Notation for the SentiWordNet database can be viewed in Appendix E.
Class Diagram of SentiWordNet class can be viewed in Appendix D.
Sequence Diagram of SentiWordNet class can be viewed in Appendix D.

The ER Notation for SentiStrength wordlist database can be viewed in Appendix E.
The generation of SentiStrength wordlist database was constructed using a csv importation tool supplied by Navicat database manager.

6.5.1 Implementation Problems

6.5.1.2 Wordlist Compatibility

SentiWordNet is essentially a clone of WordNet with sentiment attached to each of the words. During implementation SentiWordNet promptly became unusable due to the irrelevant amount of sentiment tagged words. Because SentiWordNet uses WordNet as a foundation, it is a wordlist not just containing emotional expression, but anything that may have a subtle hint of polarity.

Example Words in SentiWordNet with Polarity
Table 7

Example Words in SentiWordNet	Term	Polarity
Good	Agreeable or pleasing; "we all had a good time"; "good manners" ;	Positive: 0.625 Objective: 0.0125 Negative: 0.0
Bad	Below average in quality or performance; "a bad chess player";	Positive: 0.0 Objective: 0.25 Negative: 0.75
Rain	Precipitate as rain; "If it rains much more, we can expect some flooding"	Positive: 0.0 Objective: 0.875 Negative: 0.125
Transplant	Be transplantable; "These delicate plants do not transplant easily"	Positive: 0.25 Objective: 0.75 Negative: 0.0

As one can see, there are words that are perceived to contain polarity even though in the common sentence it is not the case. This is the first reason why SentiWordNet is not appropriate, a large quantity of words don't really contain polarity and instead skew SMME's results. A neutral tweet such as "*I just had a transplant*" could be processed as positive! Even on the assumption SentiWordNet should be used because of its detailed repository, the decision on which polarity rating for that word to be used is computationally improbable as explained next.

Example Word in SentiWordNet and Multiple Polarity Ratings
Table 8

The table shows all the terminologies the word 'painful' can be used according to SentiWordNet.

Terminology	Polarity
causing physical or psychological pain; "worked with painful slowness"	Positive: 0.125 Objective: 0.125 Negative: 0.75
causing misery or pain or distress; "it was a sore trial to him";	Positive: 0.0 Objective: 0.125 Negative: 0.875
exceptionally bad or displeasing; "atrocious taste";	Positive: 0.0 Objective: 0.125 Negative: 0.875
causing physical discomfort; "bites of black flies are more than irritating; they can be very painful"	Positive: 0.0 Objective: 0.375 Negative: 0.625

The fact each word has multiple terminologies brings on the difficulty of choosing what one to use computationally. This is the second reason SentiWordNet is impractical for SMME. The most straightforward solution is to calculate the average polarity but as studies show, this does not produce accurate results (see "Comparison Evaluation"). In conclusion, implementation forced research into an alternative wordlist.

The alternative wordlist (SentiStrength) consists of emotional words only. Each word has a rating next to it between -5 and 5, this provides easy determination what exactly is positive or negative and to what degree.

Example Words in Alternative WordList with Polarity
Table 9

Please note a large amount of words are stemmed into their base form.

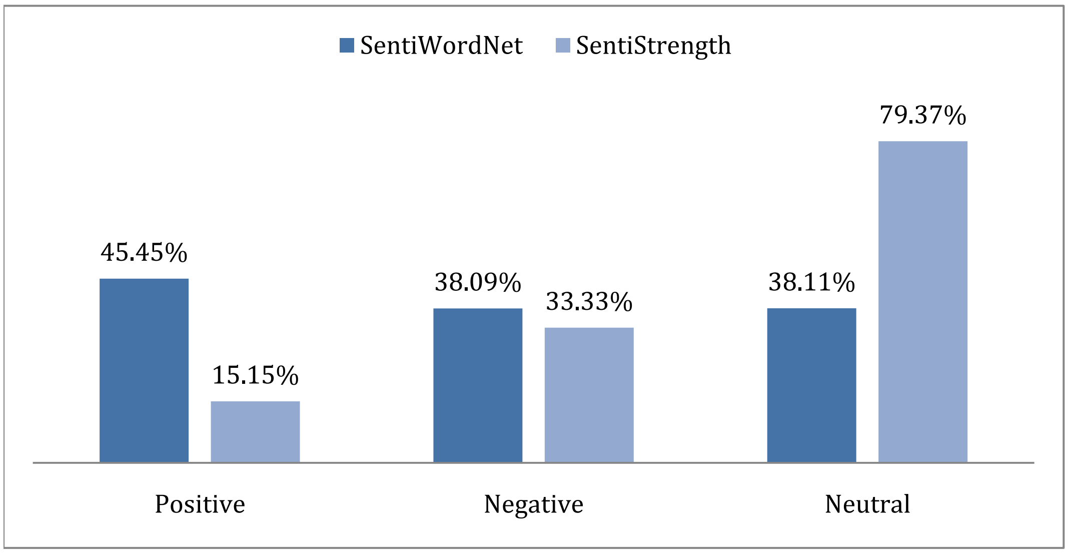
Example Words in Alternative Wordlist	Polarity
Cute*	2
Cynic*	-2
Damag*	-1
Damn*	-1
Danger*	-1

6.5.1.3 Comparison Evaluation

SentiWordNet and SentiStrength wordlist are compared to assess which wordlist is the best. The best wordlist is the one that improves overall sentiment analysis accuracy.

Wordlist Comparative Test Results: SentiWordNet VS SentiStrength

The most appropriate wordlist is calculated by using SMME to processes 500 tweets already marked into neutral, positive and negative categories by a human. The classifier alters the wordlist it uses and for each run the quantity of tweets it accurately processes is calculated as a percentage (indicating accuracy of sentiment analysis).

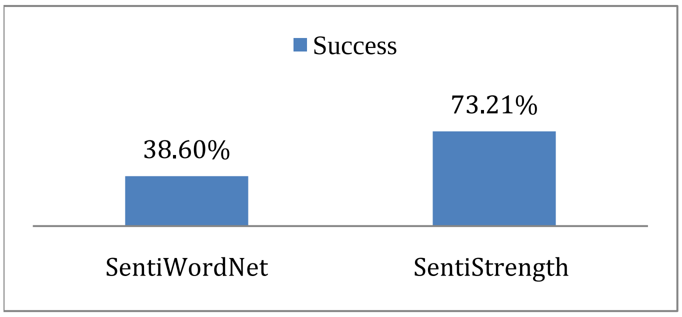


The results show that SentiWordNet provides a better sentiment accuracy than SentiStrength when it comes to positive and negative classifying, but it is 30 percent less accurate for neutral tweets. One may regard the classifying of positive and negative tweets more important and so SentiWordNet is better, but research into the total accuracy and error rate provide a different observation.

Wordlist Comparative Test Results: SentiWordNet VS SentiStrength

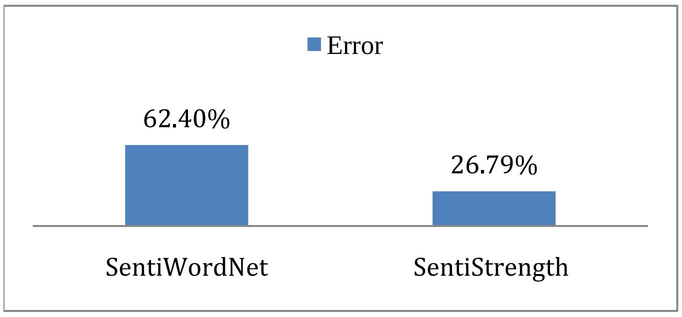
Overall Success Rate of Assessing Polarity in Tweets

The overall success rate is the total percentage of how many tweets the classifier processed correctly when using different wordlists.



Overall Percentage of Errors in Assessing Polarity of Tweets

The overall success rate is the total percentage of how many tweets the classifier processed incorrectly when using different wordlists.



The overall statistics show that SentiWordNet performs notably worse than the SentiStrength wordlist. The percentage of success is 35 percent less and 36 percent more errors. This is a significant part of implementation that changed the design (from using SentiWordNet) and one that potentially improved the overall accuracy of SMME in the 30-40 percent range.

The main reason for this dramatic difference in accuracy is because of the detail of SentiWordNet. As explained earlier, it often contains words that do not usually hold sentiment in most cases (see ‘transplant’) but the average polarity of said word according to SentiWordNet (in this example) is positive. SMME unfortunately does not need or use the amount of detail and complexity that SentiWordNet provides.

6.5.2 Testing

None.

6.6 Website

The changes to website design can be viewed in Appendix F. The change happened during implementation and is due to the language used (what is possible), user interaction improvement and general aesthetic preferences.

6.6.1 Implementation Problems

Between the initial report and current design the website has changed dramatically. The languages proposed (as documented inside the logbooks) have changed from Flash, to Ruby and finally Java. The reason for these changes initiated with potential, and finalized due to current knowledge. Java was chosen in the end, as it is a language already known. Aside from languages, multiple small changes have been adopted due to implementation issues.

- Removal of Ajax and JSP rags.
  - Ajax and JSP tags were looked at to provide a better report web service integration. This was dropped due to time resources spent trying to learn them. In the end a servlet was used to redirect a user to a new page using JavaScript for graphical data and session variables.
- Servlet
  - The use of a servlet as mentioned primarily came about due to complexity of using Ajax and JSP tags. The following issues required resolving during implementation:
    - Generating one corpus hashmap for all concurrent users. To accomplish this it is only generated if it is null (after server reset).
    - Creating a dynamic report. This was solved using session variables and redirect the user to a report.jsp page. The report page uses JSP variables to acquire session variables and presents them in JavaScript.
    - The whole of SMME required reformatting. No classes could use static variables and the tweet extractor and corpus needed separating for modularity. All classes needed changing so they could be called by object methods (constructer) rather than using a main method.

6.6.2 Testing

A combination of white and black box testing was performed to make sure the website is user friendly, prone to breaking and allowed concurrent connections.

7 Evaluation

The evaluation chapter analyses the final product of SMME, does it cater to the requirements defined and how efficient is it. More importantly, does it qualify as a sentiment analysis engine?

The evolution of design through extensive research and implementation problems has amounted to the final production of SMME. It is a system prioritized to supply academic and commercial needs for autonomous social media opinion mining; the question is, has it been a success? This can only be concluded through scientific evaluation.

7.1 Evaluation of Requirements

7.1.1 Requirements

Completed Requirements:

- Allow queries on people, places or items.
- Retrieve information from social media repository dependent on the query given.
- Create a working polarity assessment engine that produces non-random results.
- The polarity assessments engine to use a method other than simple word polarity occurrence checking to work out sentiment.
- Seventy percent or greater accuracy in polarity assessment. This means, out of all the tweets processed with sentiment analysis, seventy percent or more must correctly be tagged with a positive or negative opinion.
- Present findings in a graphical report format.
- Achieve a turn-around speed of less than ten seconds.
- Host on a local server.
- Provide an interface according to web standards and effective UI.
- Code must have effective error handling without causing crashes.
- Allow concurrency between multiple users .
- Display changes of polarity over the course of a chronological timeline.
- Report reoccurring discussed topics within the given search query

7.1.2 Failed Requirements:

- Produce a seventy percent accuracy or greater in assessing context ambiguity. This means, out of all the tweets processed for polarity, seventy percent or more of tweets that contain context ambiguity must still output the correct polarity.
- All processed tweets that have geographic data attached to their profiles the polarity is displayed appropriately on a map.
- All tweets that have gender data attached to their profiles are reported appropriately.
- Host on a server to be used from any location.

The failure of these results is not deterministic of SMME's ability or achievement. Context ambiguity is near impossible, geographic data and gender data attachment is a possible future cosmetic extension while remote hosting has no scientific or end product advantage.

The fulfillment of requirements indicates that the product in general is a success in terms of project scope. It is deliverable as a commercial application and provides answers to many of the questions asked of it in the demographic of economics and business.

7.1.3 Disputable Requirements:

- Seventy percent or greater accuracy in polarity assessment. This means, out of all the tweets processed with sentiment analysis, seventy percent or more must correctly be tagged with a positive, negative or neutral opinion.

The disputable requirement has been the main challenge of SMME, to create a sentiment analysis engine that is accurate seventy percent of the time without using a pre-existing technique. This requirement encapsulates 95 percent of the designs and implementation of SMME, to produce an accurate sentiment analysis engine. It is this requirement that distinguished SMME as not only a product to be used for specific reasons but rather a research and academic project, something to push NLE boundaries and discover something new. The question that arises is, is this disputable requirement a success or a failure?

7.2 Sentiment Analysis Performance

To evaluate SMME accuracy a test case of 500 tweets were extracted from Twitter using the keyword ‘iphone’. The keyword was used as it is something that can’t be used in a derogatory manner in order to reduce context ambiguity and because it’s a popular tweeted subject. A human marked the 500 tweets into neutral, positive and negative categories. In order for the classifier to pass the disputable requirement it will need to automatically mark the 500 tweets in their human assessed category at a rate of 70 percent accuracy.

The test case used can be found on the submitted CD under “DB Files”.

Experiment 1: Evaluating SMME Sentiment Analysis Accuracy

SMME sentiment analysis is a product derived of a classifier and corpus, the design of both of these components means it is possible to produce multiple permutations of them. The purpose of this experiment is first to determine what combination of classifier and corpus build offers the best results.

The test uses a 500 tweet test case on the term 'iphone'. A human has marked this test case into positive, negative and neutral categories. To assess the accuracy, the quantity of tweets SMME marks correctly is converted into a percentage.

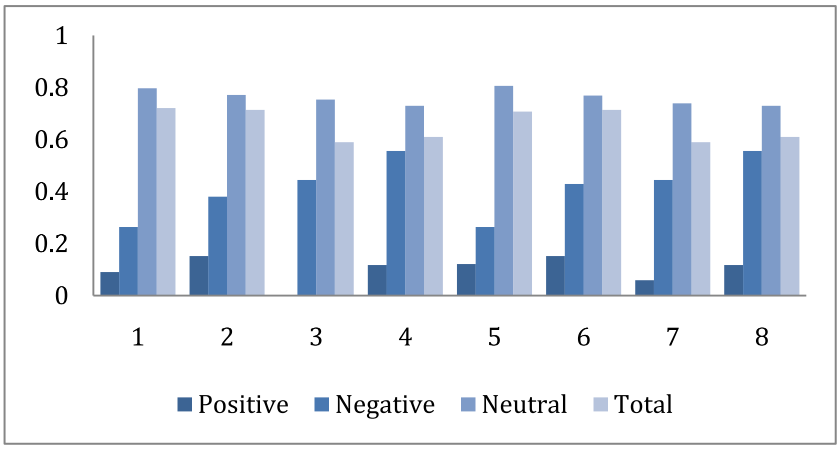
The two questions this will answer is:

- What is the best permutation of classifier and corpus design?
- Is SMME able to perform sentiment analysis with 70 percent or greater accuracy?

SMME Combinations:

Default Corpus:	Made from 10,000 tweets extracted from Twitter
Emotional Corpus:	Made from 10,000 tweets extracted from Twitter that have emoticons in.
Sentence Split:	The Sentence Split method (see Automated Corpus Gen – Ambiguity) used when classifying.
Filtering:	Filtering of spam when classifying.

Test #	Corpus	Filtering	Sentence Split	Positive	Negative	Neutral	Total
1	Default	No	Yes	9.09%	26.31%	79.67%	72.07%
2	Default	No	No	15.15%	38.09%	77.13%	71.39%
3	Default	Yes	Yes	0%	44.44%	75.36%	58.94%
4	Default	Yes	No	11.76%	55.55%	72.97%	61.00%
5	Emotional	No	Yes	12.12%	26.31%	80.60%	70.80%
6	Emotional	No	No	15.15%	42.85%	76.90%	71.39%
7	Emotional	Yes	Yes	5.88%	44.44%	73.91%	58.94%
8	Emotional	Yes	No	11.76%	55.55%	72.97%	61.00%



The experiment informs us that there is not much difference between using a default or emotional corpus in overall accuracy; however, when looking at precision recall (focusing on the positive and negative) the emotional corpus has the edge. The reason for this is probably due to emotional tweets are often subjective, building a corpus from these tweets may increase subjective training data. It confirms that the best setting for SMME is the combination of test six. These settings are used for the remainder of evaluation tests and presentation.

The other purpose this experiment serves is evidence of the disputable requirement, achieving a sentiment analysis accuracy rating of 70 percent. The experiment clearly shows that SMME achieves this, boasting over 70 percent in neutral deterministic accuracy and achieving over 70 percent on total accuracy in certain test cases. Given the fact test six is the newly adopted set up for SMME the disputable requirement is now confirmed.

7.3 Comparative Evaluation

Due to the complexity involved with NLE and sentiment processing, SMME could be classified more as a research academic project. One personal requirement throughout development was to see how well the SMME classifier and corpus compares to alternative solutions. To evaluate the potential of SMME, two other classifiers were constructed to conduct comparative analysis. A rule based **Wordcount** classifier and the statistical **Naïve Bayes** classifier.

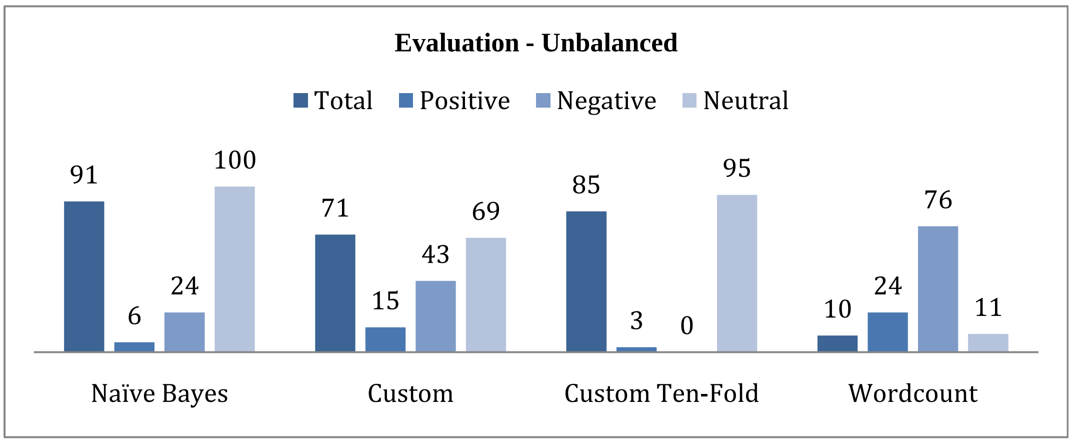
Experiment 2: Comparative Analysis

The Wordcount classifier uses the alternative wordlist to calculate any positive or negative words it finds, the one it finds the most of is used as the tweets polarity. Assuming there are no positive or negative words (or an equal amount) the tweet is classed as neutral.

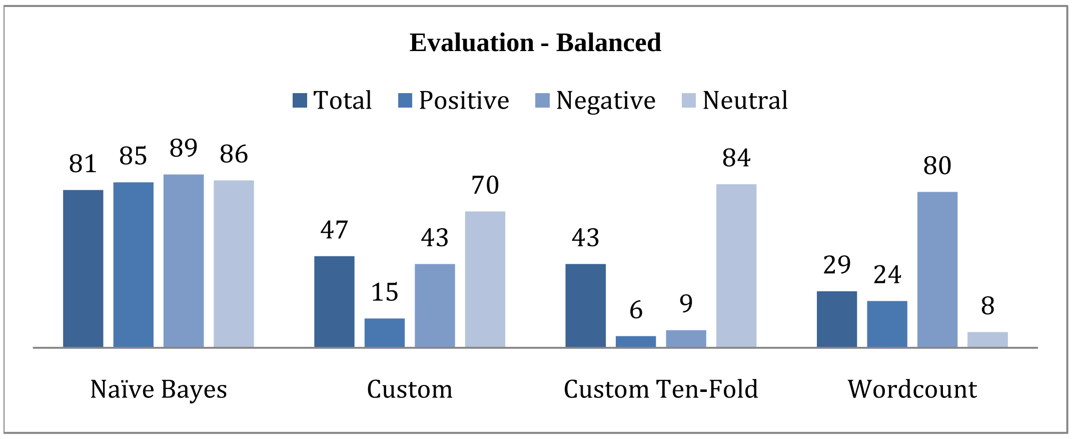
The Naïve Bayes classifier uses a corpus of bigrams to be trained on. It determines the polarity of a tweet by calculating the positive, negative and neutral probability of each word inside the tweet.

To make this experiment plausible and authentic, the Ten Cross Fold validation technique is applied. 500 human marked tweets (33 positive, 22 negative and 445 neutral) are used as the test data for this experiment.

- Naive Bayes requires a corpus. To acquire the corpus the test data is also used to build the corpus using ten cross fold. 90 percent of the test data is used for the corpus and the other 10 percent as test data, this is cycled ten times so all permutations of data is covered.
- The Custom classifier uses a 1000 word emotional corpus as the settings defined in Experiment 1.
- To gather a control comparison to Naïve Bayes, the Custom Ten-Fold classifier uses the ten cross fold technique.
- The Wordcount classifier doesn't require a corpus.



The same experiment was conducted but this time a 'balanced' test case was used. It is exactly the same test case in the previous experiment but this time has been trimmed down to 100 tweets so it contains roughly the same amount of positive, negative and neutral polarity tweets (33 positive, 22 negative, 45 neutral).



The experiment indicates that Naïve Bayes requires some form of scaling probability to achieve a higher accuracy, without this the custom classifier is far more accurate. The very low accuracy rate on the positive and negative for Custom Ten-Fold is most likely because there is not enough data to produce an accurate corpus from.

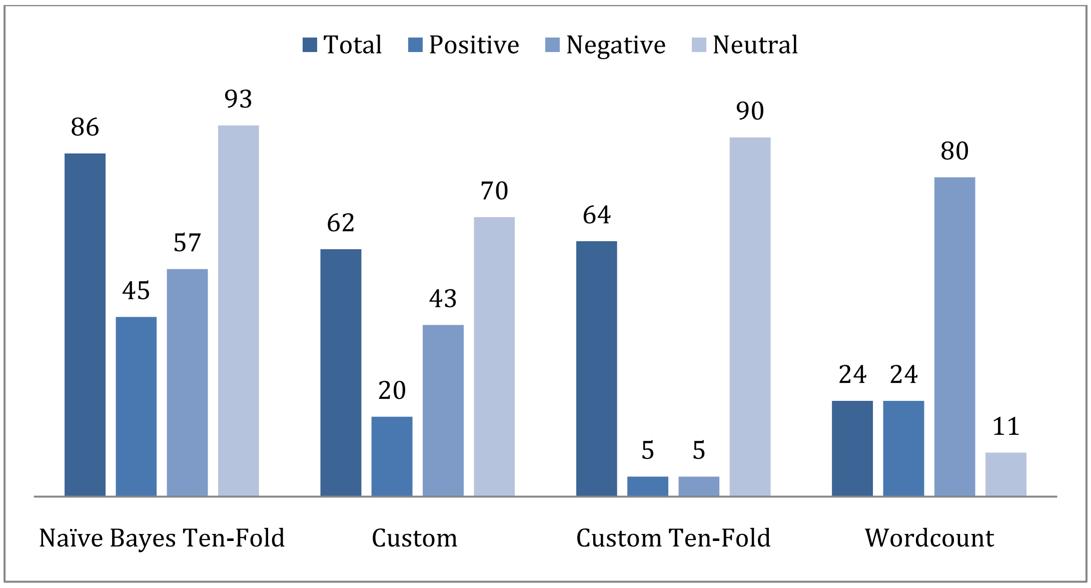
The evidence to support this is:

- The custom classifier that uses a pre-defined (immutable) corpus, the results stay consistent.
- The custom Ten-Fold test shows an improvement when building a corpus on a smaller, less noise, confined test case.

The word count classifier is exceptionally poor, although the negative ratio is extremely high this is primarily due to chance. The word count classifier usually marks a tweet as subjective even though it is not, which explains why the neutral is very low.

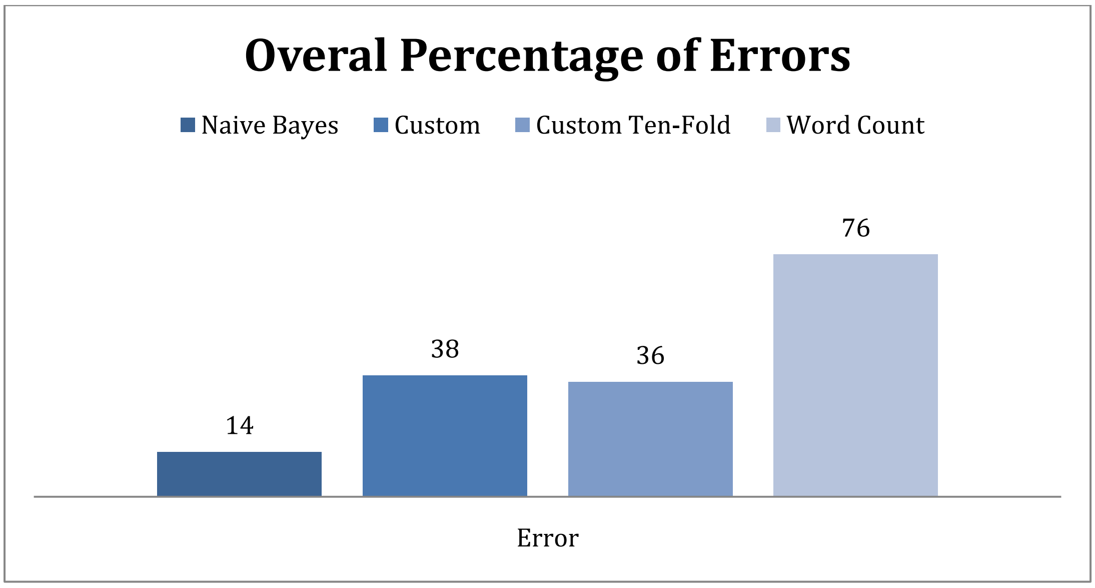
<u>Graph Displaying Classifier Aggregated Evaluation</u>
Graph 1

The results can be aggregated together for a more overall assessment of their abilities. This is the average values of both balanced and unbalanced scores.



Graph Displaying Classifier Error Ratio
Graph 2

This graph puts further emphasis on the amount of errors each classifier type makes. This further confirms that the Wordcount classifier is the worst.



The comparative study shows that Naïve Bayes is still the best classifier for sentiment analysis. The classifier SMME uses however is still acceptable, on average only being 25 percent less accurate. This is a remarkable achievement for a completely original statistical classifier and corpus design developed in six months and all completely automated (no need for a human defined corpus).

7.4 SMME Classifier Evaluation

One question that is irrelevant to any requirements:

What is stopping the classifier becoming greater than Naïve Bayes and becoming 100 percent accurate?

Naïve Bayes and SMME classifier determine polarity in text using different approaches. Naïve Bayes does not care about the context (sentiment) of words, instead it relies on the complete probability of occurrence. SMME on the other hand tries to determine which word inside the text holds the sentiment in reference to the searched term. This technique is very flawed in regards to text that doesn't contain a word with polarity but sentiment is implied.

Example:

Tweet: Going to shops to finally get the iPhone!! I've been waiting forever for this day

This leads onto the fundamental issues with all SA practices and why at the present time, no classifier is achieving 100 percent accuracy: ambiguity. Ambiguity can be present in many forms from sarcasm to broken English. Ambiguity is such a difficult task to compute, even humans have difficulty with it. The test case of 500 tweets used throughout this projects evaluation experiments was assessed by person A and the tweets were placed into positive, negative and neutral categories. The same test case was marked by someone else, person B who remarkably made fifteen amendments to person A! The truth is, every person perceives text differently and as a result sentiment analysis will most likely never become 100 percent accurate.

8. Project Management / Lessons Learnt

The project management chapter overviews the skills used and learnt to keep SMME on track.

The project was managed via the use of a Gantt chart. The chart originated from the initial report and as one can imagine, the elements of said chart rapidly changed throughout development due to design changes, implementation flaws, general poor-time management and other disregarded university assignments.

The initial, interim and final Gantt charts can be seen in Appendix G, H and I.

The predominant lessons learnt that are evident in the Gantt charts progression are:

- A smaller less ambitious project
- Perform more research

The feedback mentioned in the interim report deliberated that the project is over-ambitious. After countless hours of research, designing and implementation it would seem this statement to be factual. Yes, the project is an over-ambitious subject for an undergraduate to tackle and something easier could potentially achieve the same marks – but the question is, would this satisfy the enthusiasm and curiosity at a personal level? The statement does however encourage one to ponder the time resource to result ratio. If the project could be reproduced, a more direct focus on a given area would be concentrated on (IE: The research potential of SMME as a sentiment classifier rather than trying to morph it into a product). This would produce a more refined result rather than trying to spread the resources available too far.

Researching the field of sentiment analysis with no prior background became an issue. The little knowledge of the NLE domain left potential academic influences redundant and not investigated further. A prime example of is this after using Dr. Turney's work to automate the generation of the corpus, a different paper expression similar (potentially more relevant) research on the subject [ref] was found by luck. The largest error occurring from the lack of prior knowledge is not really knowing the problems that would emerge with ambiguity, complexities with using syntactical focus (rather than 'bag of words' technique) and managing time to reflect the many 'rookie' misunderstandings/mistakes. This is of course is expected with a project within a newly explored domain. In future work broader research is a requirement as it is all too easy to become infatuated with a specific idea, which often leads to tunnel vision. This also holds true with something as simple as implementation, it is all too easy to ignore the complexity of the concept in design until the actual production is required.

8.1 SVN Logs

As evidence of steady (non-conflicting) work and backup purposes, an SVN host was used throughout the project development.

The SVN logs can be viewed inside Appendix H.

Please note, there are significant amounts of missing commitments. This is simply because on multiple occasions commitment of work was forgotten.

9. Conclusions and Future Work

To conclude…

SMME managed to achieve fourteen out of eighteen requirements, but does this make SMME a failed development? The answer is no. Requirements are immutable defined steps made at the beginning of project production in the initial report. During the project evolution into its current state there have been changes not originally perceived to happen when the project was first initiated. The fact that SMME fulfills enough goals to become an independent piece of software and is commercially viable, it is in own right a success. The most important factor for SMME is the requirement of obtaining over 70 percent in sentiment analysis – this alone defined SMME as a project, one that became more academic based than a sellable, usable, commercial product.

The development of SMME over the past six months has evolved into a project that pushes the boundaries of natural language engineering. The designs from automated corpus generation to classifier are completely new techniques developed by myself in the allotted timeframe. Considering the accuracy gained (see evaluation) compared to the Naïve Bayes classifier, a highly human involved model derived from the Bayes Theorem shows SMME has significant potential as a product and also as a piece of research.

To conclude: is SMME a success? Yes it is.

9.1 Future Work

Natural language engineering and sentiment analysis are very broad domains in the computer science realm. The project only reflected a tiny part of what is currently being researched for computers to take the next step in artificial intelligence. SMME has many possibilities of future work ranging from enhancing SMME as a sentiment analysis engine to extending it for a different use in the NLE field. The engine could easily be adopted into a spam filter or used to perform SA on other data (social networking, rss feeds, blogs, reviews etc). If not for future development, the engine and design plans could be dissected for research purposes in the academic field to perhaps aid as future influence.

In terms of product enrichment, the engine could be marginally improved by consistent tests and tweaks. If time was permitting this is something that would be high on the priority list. It was lightly touched on in the evaluation chapter about the quantity of permutations of techniques available to SMME processing. The results from these tests showed dramatic changes in results (up to 23%). There are plenty of obligatory components that also affect this rate such as the tokenizing technique, part of speech tagger, or even the trigrams used to extract tweets with polarity could potentially improve accuracy by a further 5-10%. There is little doubt in my mind the accuracy of SMME could be heightened by the simple changes but unfortunately requires an investment of time, something that is restricted at present.

The embodiment of SMME is not deceased; it holds great potential to compete against the other commercial Twitter sentiment engines, or lie amongst future influential work as a new type of statistical classifier.

10. References

Xu, Z. (2010). *A Sentiment Analysis Model Integrating Multiple Algorithms and Diverse Features.* Ohio: The Ohio State University.

Yusuke. (2007, 06 09). *Twitter4J: Twitter API for Java*. Retrieved 01 02, 2011, from Twitter4J: http://twitter4j.org/en/index.html

Apache, T. (1999). *Apache Tomcat*. Retrieved 11 12, 2010, from Apache Tomcat Project: http://tomcat.apache.org/

Baccianella, S., Esuli, A., & Sebastiani, F. (2010). *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.* Pisa, Italy: Istituto di Scienza e Tecnologie dell'Informazion.

Beck, K. (2009, 27 05). *JUnit.org Resources for Tesr Driven Development*. Retrieved 12 01, 2010, from JUnit: http://www.junit.org/

Go, A., Bhayani, R., & Huang, L. (2009). *Twitter Sentiment Classification using Distance Supervision.* Stanford, California, USA: Stanford University.

Ingersoll, G. (2010). *Apache OpenNLP* . Retrieved 12 2010, 22, from OpenNLP: http://incubator.apache.org/opennlp/

Kerby, L. (2011). *SMME: Abstract.* England: University of Essex.

Kerby, L. *SMME: Interim Report.* Colchester: University of Essex.

Larson, D. (2010, August 11). *10 Ways and 20 Features for Searching Old Tweets*. Retrieved November 26, 2010, from TweetSmarter: http://blog.tweetsmarter.com/twitter-search/10-ways-and-20-features-for-searching-old-tweets/
Leitch, J. (2008, May). *Google: Json*. Retrieved 2 02, 2011, from A Java Library to Convert JSON to Java Objects and Vice-Versa: http://code.google.com/p/google-gson/

Mason, O. (2010). *NLE QTagger*. Retrieved 11 26, 2010, from University of Birmingham: http://web.bham.ac.uk/o.mason/software/tagger/index.html

Pak, A., & Paroubek, P. (2010). *Twitter as a Corpus for Sentiment Analysis and Opinion Mining.* Orsay Cedex, France: University of Paris.

Pang, B., Lee, L., & Vaithyanathan, S. (2002, 07). Thumps up? Sentiment Classification using Machine Learning Techniques. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* , pp. 79-86.

Penner, C. (2011, 03 14). *Twitter*. Retrieved 04 18, 2011, from Twitter Blog: #numbers: http://blog.twitter.com/2011/03/numbers.html

PostGres , D. (2004). *JDBC PostGres Driver*. Retrieved 01 15, 2011, from PostGresSQL: http://jdbc.postgresql.org/

Search Engine Journal. (2010, March 03). *3 Tools to Analyze the Sentiment of Your Brand Social Mentions*. Retrieved November 17, 2010, from Search Engine Journal: http://www.searchenginejournal.com/analyze-sentiment-of-brand-social-mentions/18478/

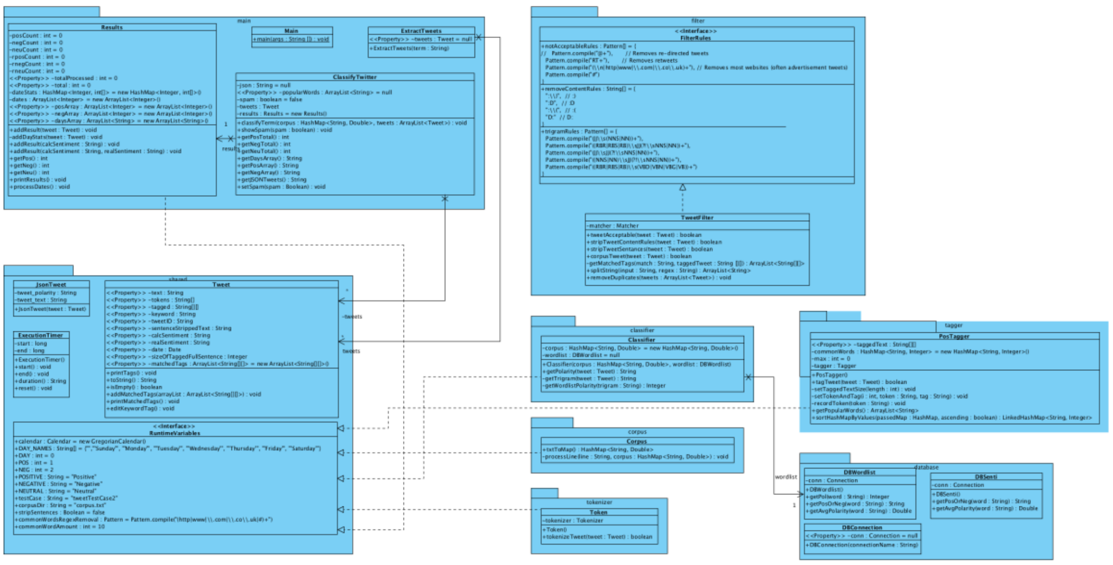
Turney, P. D. (2002, July). Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. *Proceedings of the 40th Annual Meeting of the Associations for Computational Linguistics (ACL)* , pp. 417-424.

Thelwall, M., Buckley, K., Paltoglou, G., & Cai, D. (2010). Sentiment Strength Detection in Short Informal Text. *Journal of the American Society for Information Science and Technology* , pp. 2544-2558.

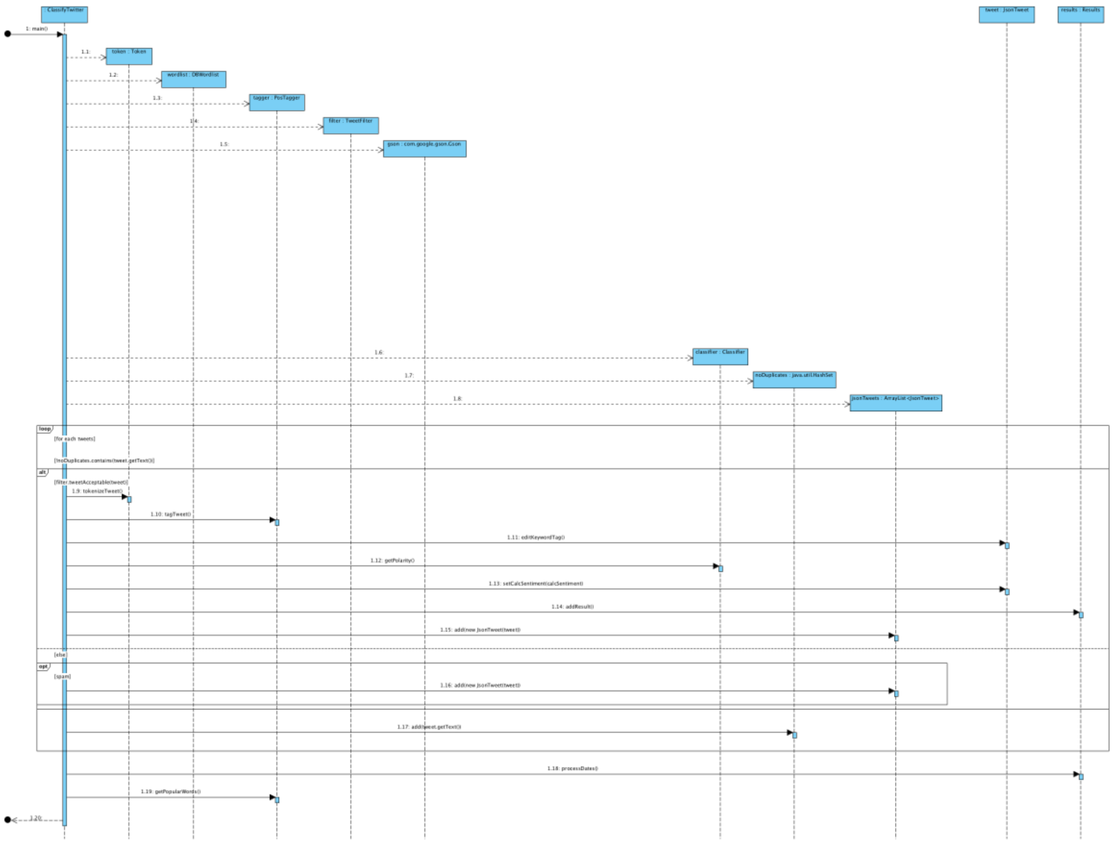
Appendix A

High definition PNG's of these diagrams can be viewed inside the folder 'SMME Diagrams' on both the submitted CD and submitted ZIP file.

Classifier Class Diagram



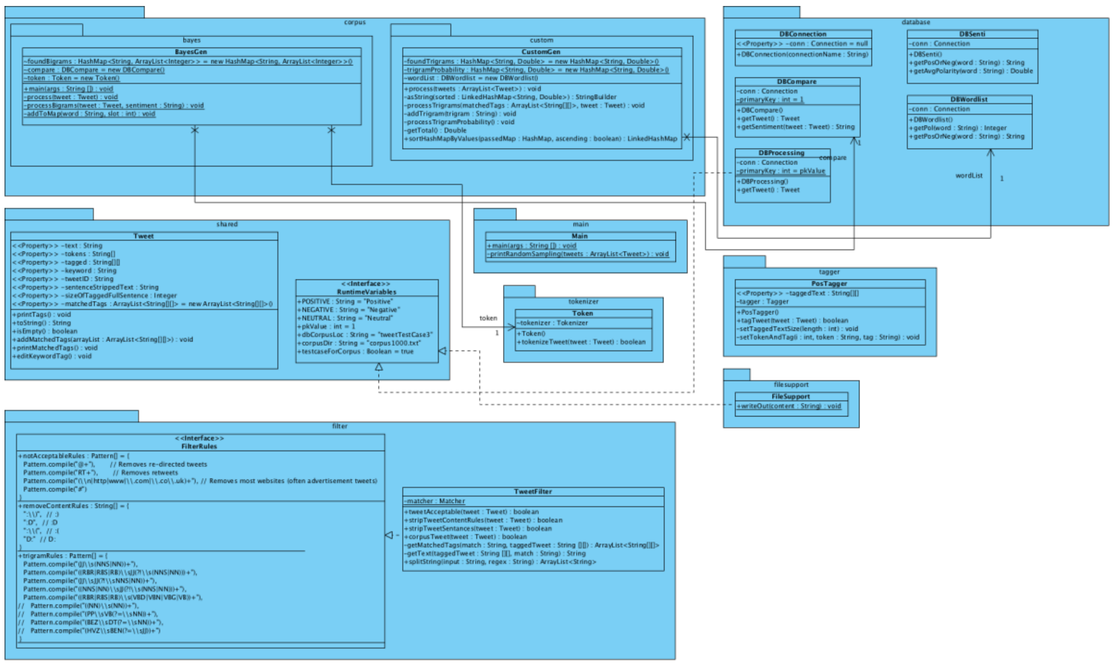
Classifier Sequence Diagram



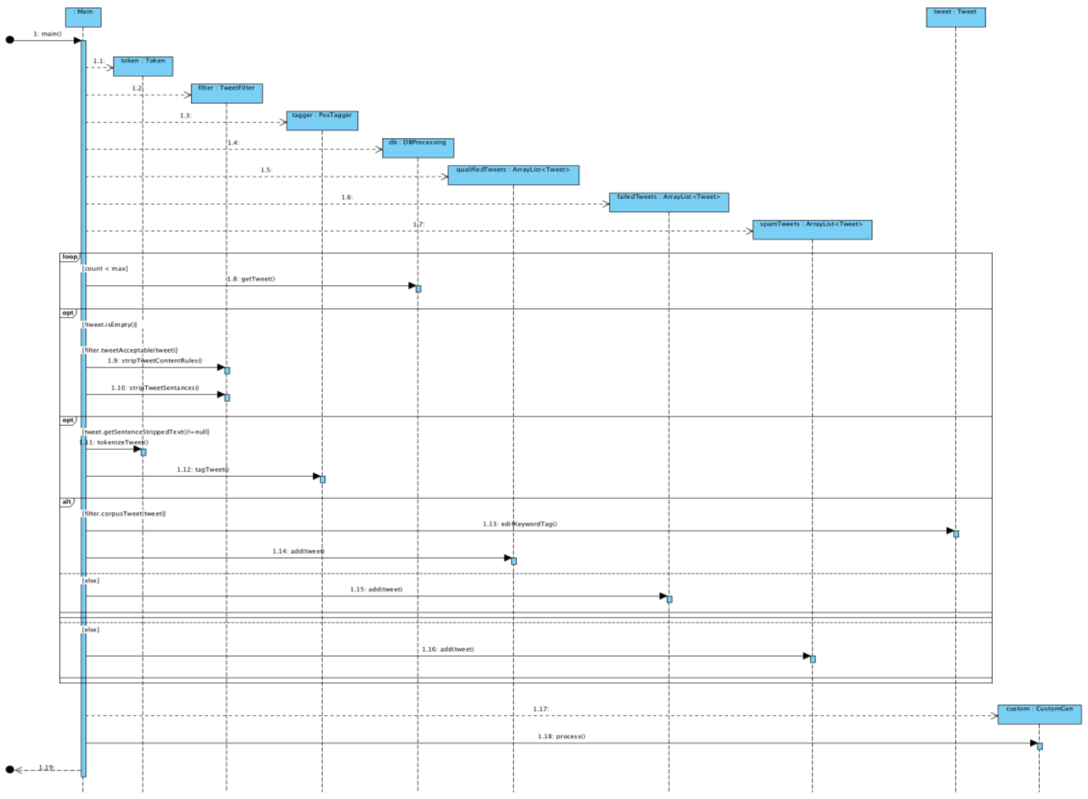
Appendix B

High definition PNG's of these diagrams can be viewed inside the folder 'SMME Diagrams' on both the submitted CD and submitted ZIP file.

Corpus Generator Class Diagram



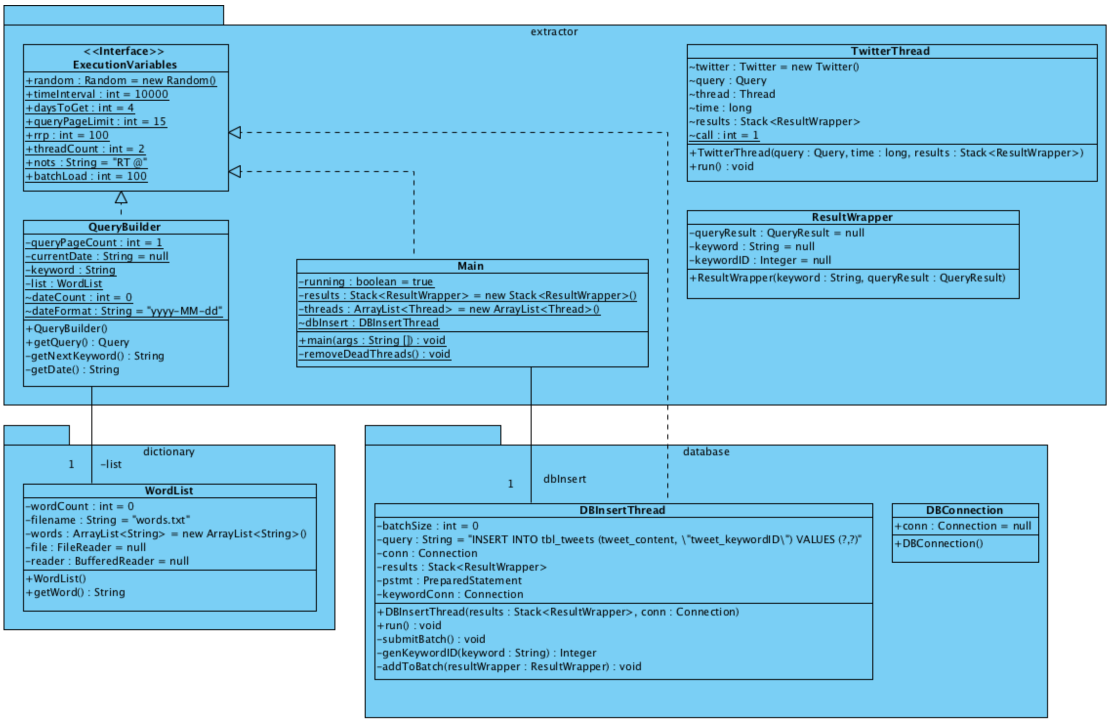
Corpus Generator Sequence Diagram



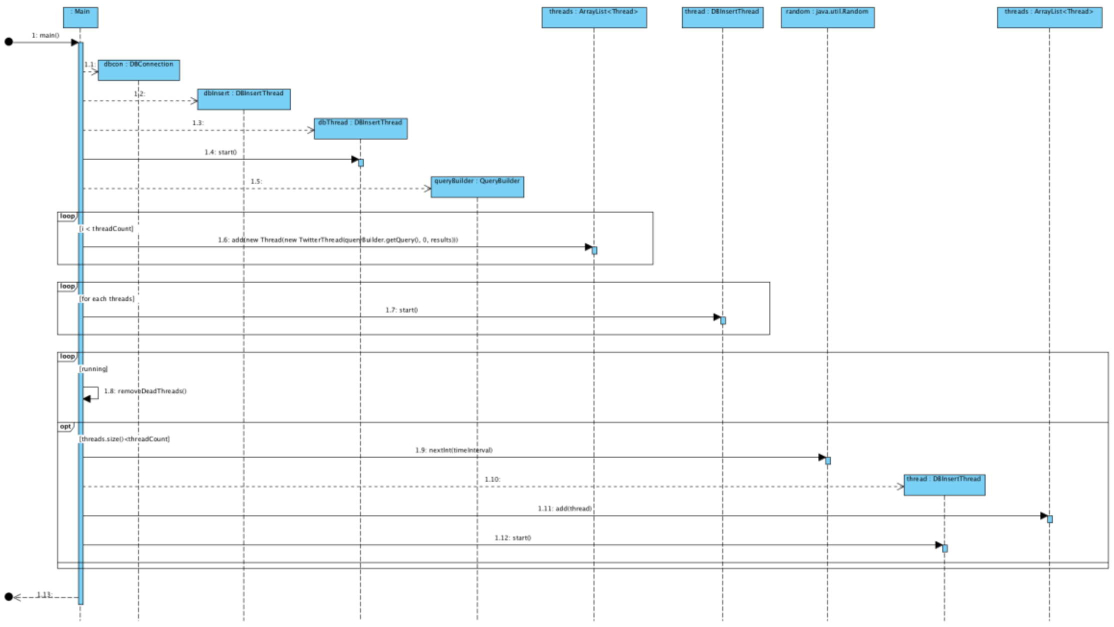
Appendix C

High definition PNG's of these diagrams can be viewed inside the folder 'SMME Diagrams' on both the submitted CD and submitted ZIP file.

Twitter Extractor Class Diagram


extractor
<<Interface>>
ExecutionVariables
+random : Random = new Random()
+timeInterval : int = 10000
+daysToGet : int = 4
+queryPageLimit : int = 15
+rrp : int = 100
+threadCount : int = 2
+nots : String = "RT @"
+batchLoad : int = 100
QueryBuilder
-queryPageCount : int = 1
-currentDate : String = null
-keyword : String
-list : WordList
-dateCount : int = 0
-dateFormat : String = "yyyy-MM-dd"
+QueryBuilder()
+getQuery() : Query
-getNextKeyword() : String
-getDate() : String
Main
-running : boolean = true
-results : Stack<ResultWrapper> = new Stack<ResultWrapper>()
-threads : ArrayList<Thread> = new ArrayList<Thread>()
-dbInsert : DBInsertThread
+main(args : String []) : void
-removeDeadThreads() : void
TwitterThread
-twitter : Twitter = new Twitter()
-query : Query
-thread : Thread
-time : long
-results : Stack<ResultWrapper>
-call : int = 1
+TwitterThread(query : Query, time : long, results : Stack<ResultWrapper>)
+run() : void
ResultWrapper
-queryResult : QueryResult = null
-keyword : String = null
-keywordID : Integer = null
+ResultWrapper(keyword : String, queryResult : QueryResult)
dictionary
1
-list
WordList
-wordCount : int = 0
-filename : String = "words.txt"
-words : ArrayList<String> = new ArrayList<String>()
-file : FileReader = null
-reader : BufferedReader = null
+WordList()
+getWord() : String
database
1
dbInsert
DBInsertThread
-batchSize : int = 0
-query : String = "INSERT INTO tbl_tweets (tweet_content, \"tweet_keywordID\") VALUES (?,?)"
-conn : Connection
-results : Stack<ResultWrapper>
-pstmt : PreparedStatement
-keywordConn : Connection
+DBInsertThread(results : Stack<ResultWrapper>, conn : Connection)
+run() : void
-submitBatch() : void
-genKeywordID(keyword : String) : Integer
-addToBatch(resultWrapper : ResultWrapper) : void
DBConnection
+conn : Connection = null
+DBConnection()

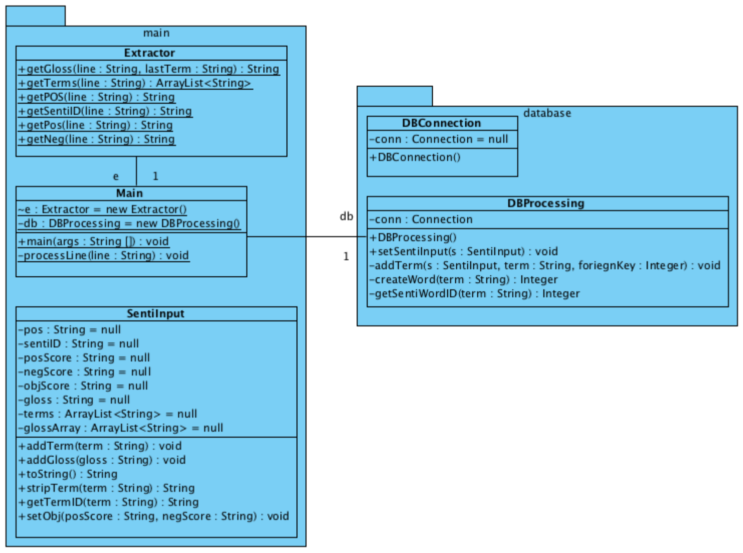
Twitter Extractor Sequence Diagram


: Main
threads : ArrayList<Thread>
thread : DBInsertThread
random : java.util.Random
threads : ArrayList<Thread>
1: main()
1.1:
dbcon : DBConnection
1.2:
dbInsert : DBInsertThread
1.3:
dbThread : DBInsertThread
1.4: start()
1.5:
queryBuilder : QueryBuilder
loop
[i < threadCount]
1.6: add(new Thread(new TwitterThread(queryBuilder.getQuery(), 0, results)))
loop
[for each threads]
1.7: start()
loop
[running]
1.8: removeDeadThreads()
opt
[threads.size()<threadCount]
1.9: nextInt(timeInterval)
1.10:
thread : DBInsertThread
1.11: add(thread)
1.12: start()
1.11:

Appendix D

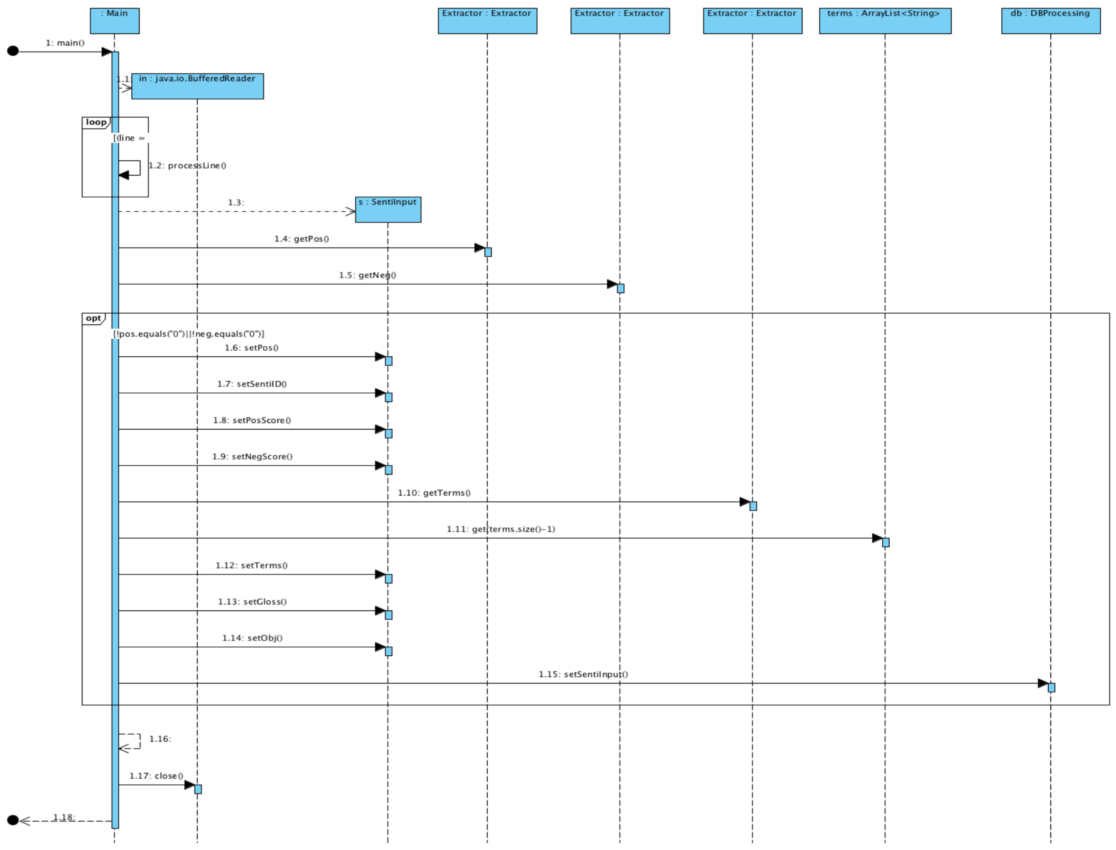
High definition PNG's of these diagrams can be viewed inside the folder 'SMME Diagrams' on both the submitted CD and submitted ZIP file.

SentiWordNet Wordlist Generator Class Diagram



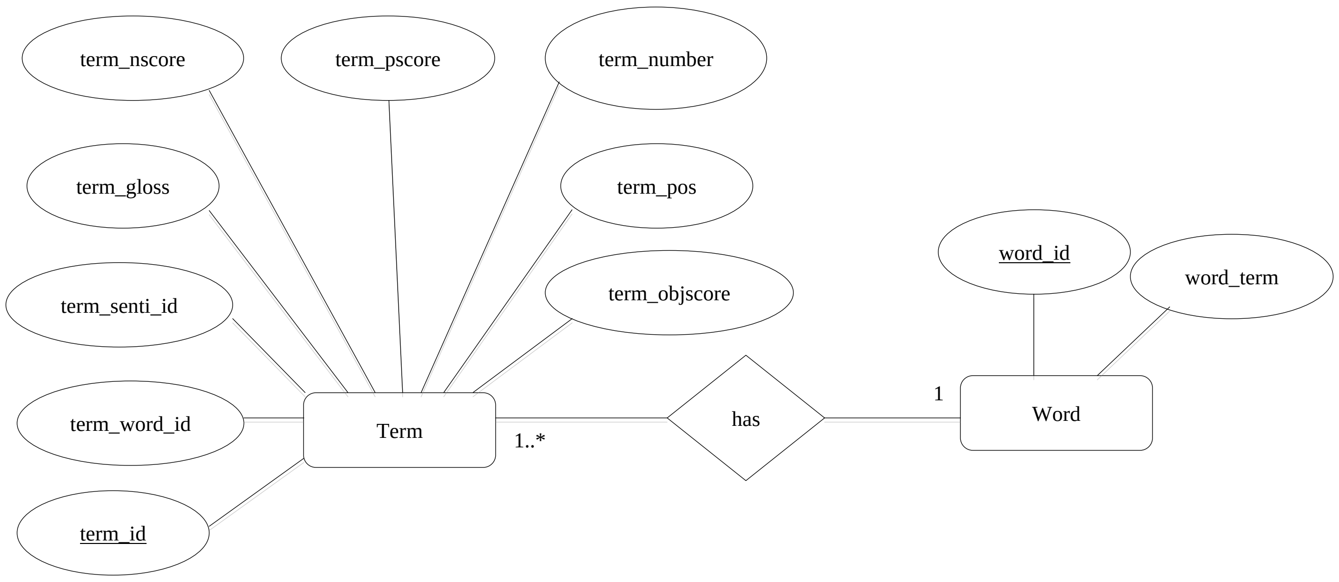
main
Extractor
+getGloss(line : String, lastTerm : String) : String
+getTerms(line : String) : ArrayList<String>
+getPOS(line : String) : String
+getSentiID(line : String) : String
+getPos(line : String) : String
+getNeg(line : String) : String
e
1
Main
~e : Extractor = new Extractor()
~db : DBProcessing = new DBProcessing()
+main(args : String []) : void
~processLine(line : String) : void
SentiInput
-pos : String = null
-sentiID : String = null
-posScore : String = null
-negScore : String = null
-objScore : String = null
-gloss : String = null
-terms : ArrayList<String> = null
-glossArray : ArrayList<String> = null
+addTerm(term : String) : void
+addGloss(gloss : String) : void
+toString() : String
+stripTerm(term : String) : String
+getTermID(term : String) : String
+setObj(posScore : String, negScore : String) : void
db
1
database
DBConnection
-conn : Connection = null
+DBConnection()
DBProcessing
-conn : Connection
+DBProcessing()
+setSentiInput(s : SentiInput) : void
-addTerm(s : SentiInput, term : String, foriegnKey : Integer) : void
-createWord(term : String) : Integer
-getSentiWordID(term : String) : Integer

SentiWordNet Wordlist Generator Sequence Diagram

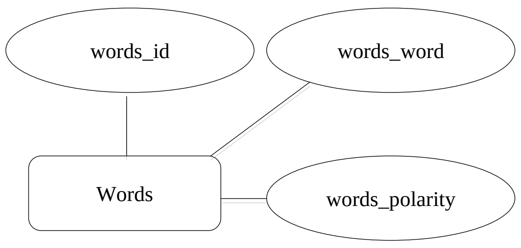


Appendix E

SentiWordNet ER Diagram

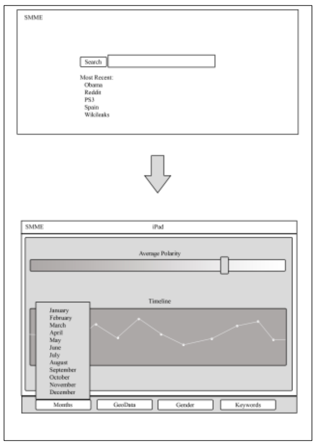


SentiStrength ER Diagram

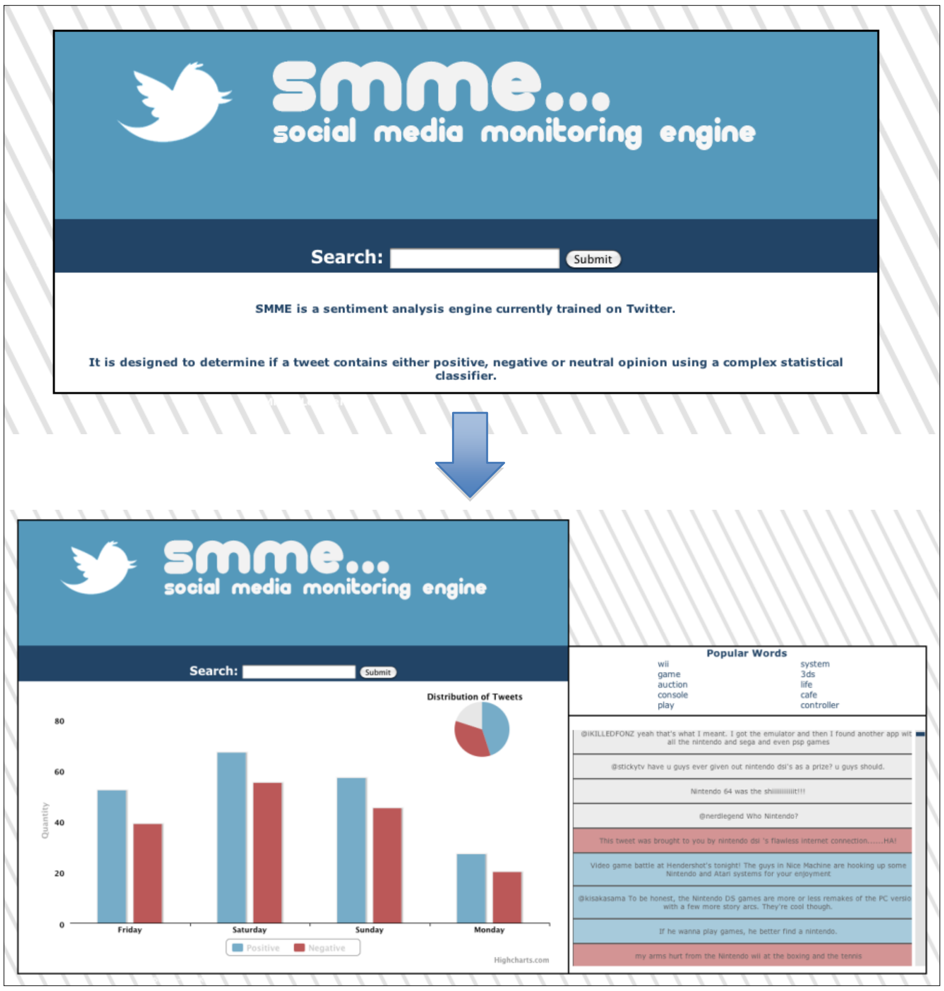


Appendix F

Website Design (Interim Report)



Website Design (Final Variation)



SMME is a sentiment analysis engine currently trained on Twitter.

It is designed to determine if a tweet contains either positive, negative or neutral opinion using a complex statistical classifier.

@iKILLEDFONZ yeah that's what I meant. I got the emulator and then I found another app wit all the nintendo and sega and even psp games

@stickytv have u guys ever given out nintendo dsi's as a prize? u guys should.

Nintendo 64 was the shiiiiiiiiiiiit!!!

@nerdlegend Who Nintendo?

This tweet was brought to you by nintendo dsi 's flawless internet connection.......HA!

Video game battle at Hendershot's tonight! The guys in Nice Machine are hooking up some Nintendo and Atari systems for your enjoyment

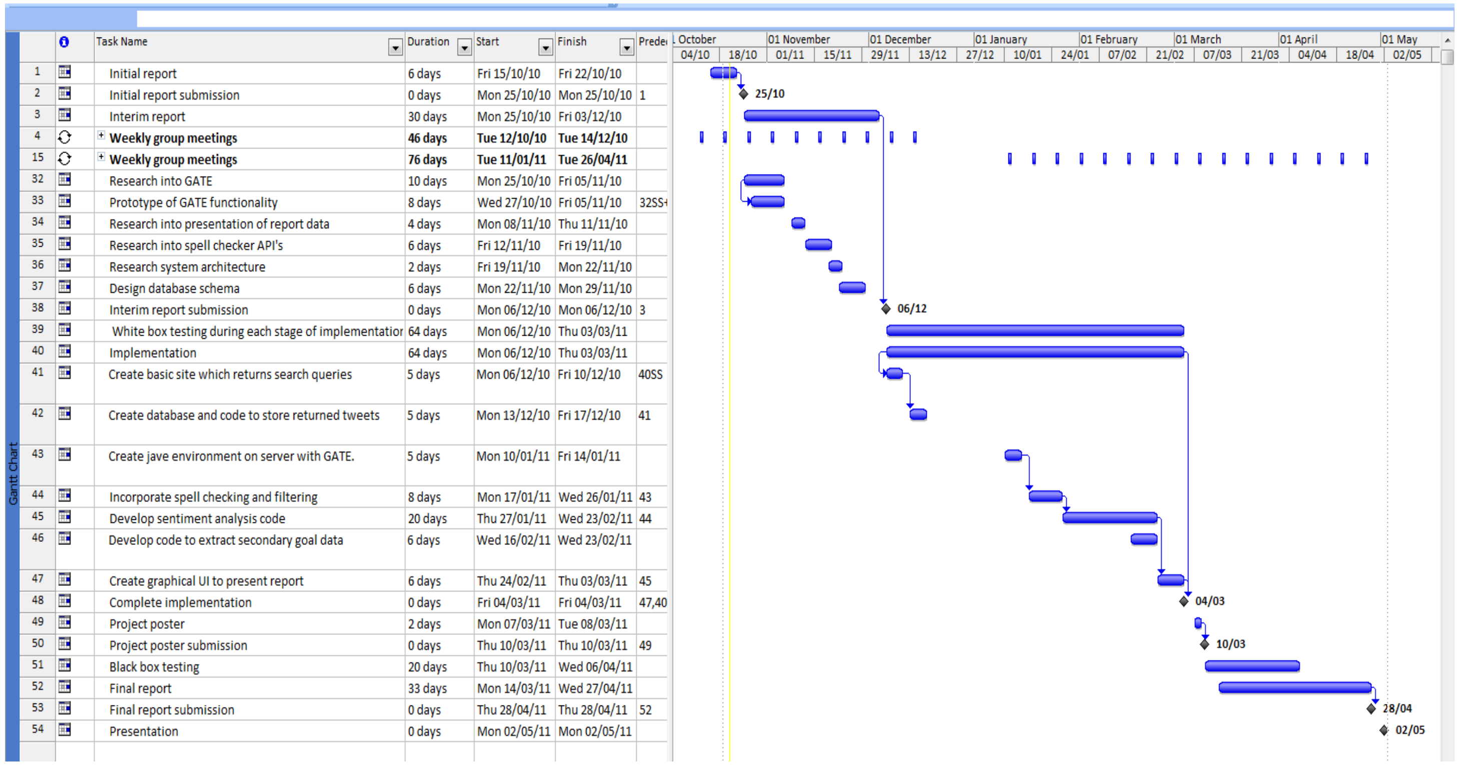
@kisakasama To be honest, the Nintendo DS games are more or less remakes of the PC versio with a few more story arcs. They're cool though.

If he wanna play games, he better find a nintendo.

my arms hurt from the Nintendo wii at the boxing and the tennis

Appendix G

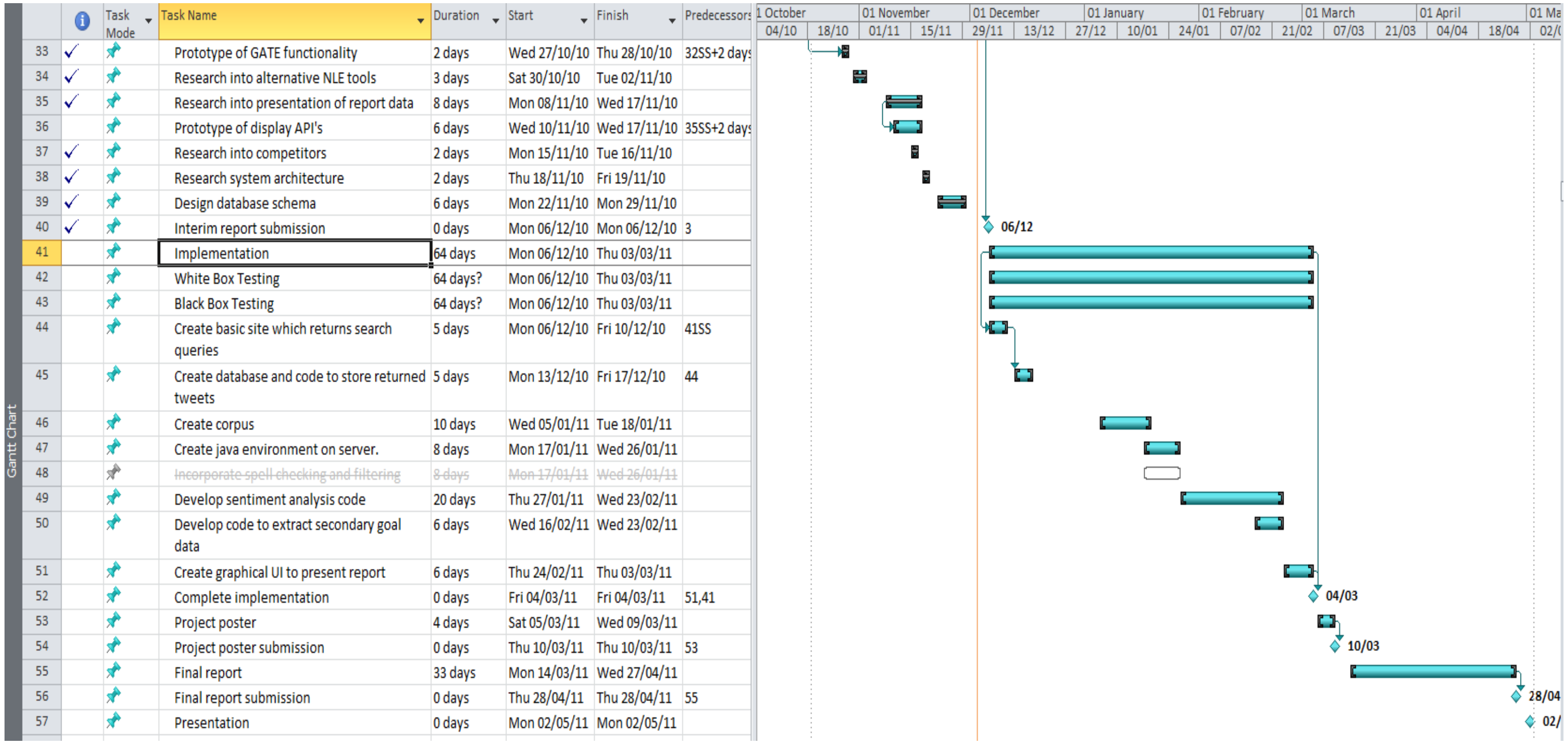
Initial Report – Gantt Chart



		Task Name	Duration	Start	Finish	Prede
1		Initial report	6 days	Fri 15/10/10	Fri 22/10/10	
2		Initial report submission	0 days	Mon 25/10/10	Mon 25/10/10	1
3		Interim report	30 days	Mon 25/10/10	Fri 03/12/10	
4		**Weekly group meetings**	**46 days**	**Tue 12/10/10**	**Tue 14/12/10**	
15		**Weekly group meetings**	**76 days**	**Tue 11/01/11**	**Tue 26/04/11**	
32		Research into GATE	10 days	Mon 25/10/10	Fri 05/11/10	
33		Prototype of GATE functionality	8 days	Wed 27/10/10	Fri 05/11/10	32SS
34		Research into presentation of report data	4 days	Mon 08/11/10	Thu 11/11/10	
35		Research into spell checker API's	6 days	Fri 12/11/10	Fri 19/11/10	
36		Research system architecture	2 days	Fri 19/11/10	Mon 22/11/10	
37		Design database schema	6 days	Mon 22/11/10	Mon 29/11/10	
38		Interim report submission	0 days	Mon 06/12/10	Mon 06/12/10	3
39		White box testing during each stage of implementatior	64 days	Mon 06/12/10	Thu 03/03/11	
40		Implementation	64 days	Mon 06/12/10	Thu 03/03/11	
41		Create basic site which returns search queries	5 days	Mon 06/12/10	Fri 10/12/10	40SS
42		Create database and code to store returned tweets	5 days	Mon 13/12/10	Fri 17/12/10	41
43		Create jave environment on server with GATE.	5 days	Mon 10/01/11	Fri 14/01/11	
44		Incorporate spell checking and filtering	8 days	Mon 17/01/11	Wed 26/01/11	43
45		Develop sentiment analysis code	20 days	Thu 27/01/11	Wed 23/02/11	44
46		Develop code to extract secondary goal data	6 days	Wed 16/02/11	Wed 23/02/11	
47		Create graphical UI to present report	6 days	Thu 24/02/11	Thu 03/03/11	45
48		Complete implementation	0 days	Fri 04/03/11	Fri 04/03/11	47,40
49		Project poster	2 days	Mon 07/03/11	Tue 08/03/11	
50		Project poster submission	0 days	Thu 10/03/11	Thu 10/03/11	49
51		Black box testing	20 days	Thu 10/03/11	Wed 06/04/11	
52		Final report	33 days	Mon 14/03/11	Wed 27/04/11	
53		Final report submission	0 days	Thu 28/04/11	Thu 28/04/11	52
54		Presentation	0 days	Mon 02/05/11	Mon 02/05/11	

Appendix H

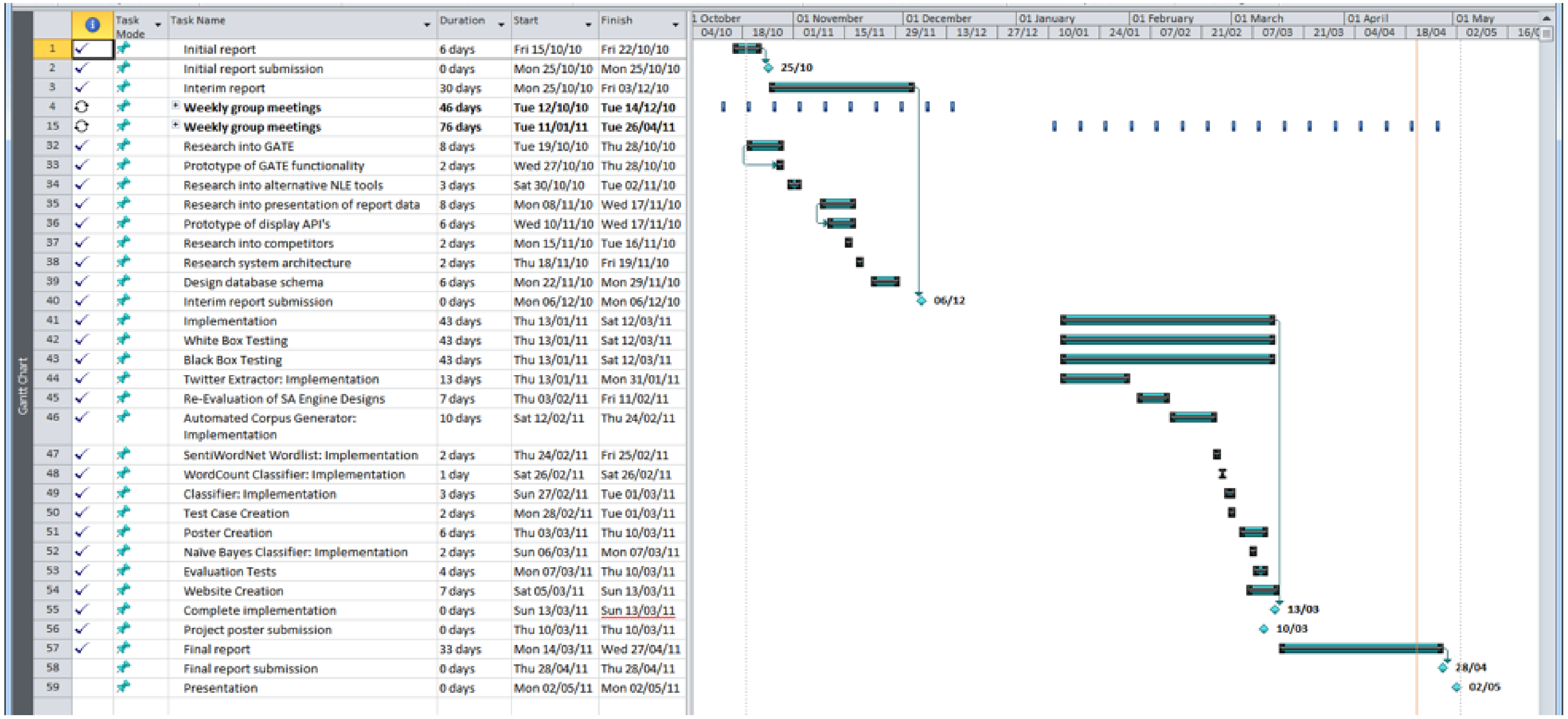
Interim Report – Gantt Chart



		Task Mode	Task Name	Duration	Start	Finish	Predecessors
33	✓		Prototype of GATE functionality	2 days	Wed 27/10/10	Thu 28/10/10	32SS+2 days
34	✓		Research into alternative NLE tools	3 days	Sat 30/10/10	Tue 02/11/10	
35	✓		Research into presentation of report data	8 days	Mon 08/11/10	Wed 17/11/10	
36			Prototype of display API's	6 days	Wed 10/11/10	Wed 17/11/10	35SS+2 days
37	✓		Research into competitors	2 days	Mon 15/11/10	Tue 16/11/10	
38	✓		Research system architecture	2 days	Thu 18/11/10	Fri 19/11/10	
39	✓		Design database schema	6 days	Mon 22/11/10	Mon 29/11/10	
40	✓		Interim report submission	0 days	Mon 06/12/10	Mon 06/12/10	3
41			Implementation	64 days	Mon 06/12/10	Thu 03/03/11	
42			White Box Testing	64 days?	Mon 06/12/10	Thu 03/03/11	
43			Black Box Testing	64 days?	Mon 06/12/10	Thu 03/03/11	
44			Create basic site which returns search queries	5 days	Mon 06/12/10	Fri 10/12/10	41SS
45			Create database and code to store returned tweets	5 days	Mon 13/12/10	Fri 17/12/10	44
46			Create corpus	10 days	Wed 05/01/11	Tue 18/01/11	
47			Create java environment on server.	8 days	Mon 17/01/11	Wed 26/01/11	
48			~~Incorporate spell checking and filtering~~	~~8 days~~	~~Mon 17/01/11~~	~~Wed 26/01/11~~	
49			Develop sentiment analysis code	20 days	Thu 27/01/11	Wed 23/02/11	
50			Develop code to extract secondary goal data	6 days	Wed 16/02/11	Wed 23/02/11	
51			Create graphical UI to present report	6 days	Thu 24/02/11	Thu 03/03/11	
52			Complete implementation	0 days	Fri 04/03/11	Fri 04/03/11	51,41
53			Project poster	4 days	Sat 05/03/11	Wed 09/03/11	
54			Project poster submission	0 days	Thu 10/03/11	Thu 10/03/11	53
55			Final report	33 days	Mon 14/03/11	Wed 27/04/11	
56			Final report submission	0 days	Thu 28/04/11	Thu 28/04/11	55
57			Presentation	0 days	Mon 02/05/11	Mon 02/05/11	

Appendix I

Final Report – Gantt Chart

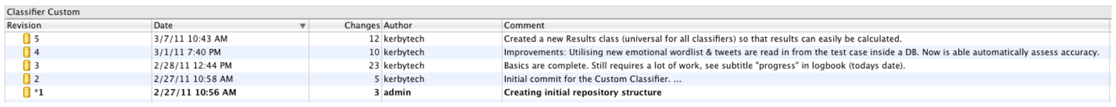


	ℹ	Task Mode	Task Name	Duration	Start	Finish
1	✓	📌	Initial report	6 days	Fri 15/10/10	Fri 22/10/10
2	✓	📌	Initial report submission	0 days	Mon 25/10/10	Mon 25/10/10
3	✓	📌	Interim report	30 days	Mon 25/10/10	Fri 03/12/10
4	⟳	📌	+ **Weekly group meetings**	**46 days**	**Tue 12/10/10**	**Tue 14/12/10**
15	⟳	📌	+ **Weekly group meetings**	**76 days**	**Tue 11/01/11**	**Tue 26/04/11**
32	✓	📌	Research into GATE	8 days	Tue 19/10/10	Thu 28/10/10
33	✓	📌	Prototype of GATE functionality	2 days	Wed 27/10/10	Thu 28/10/10
34	✓	📌	Research into alternative NLE tools	3 days	Sat 30/10/10	Tue 02/11/10
35	✓	📌	Research into presentation of report data	8 days	Mon 08/11/10	Wed 17/11/10
36	✓	📌	Prototype of display API's	6 days	Wed 10/11/10	Wed 17/11/10
37	✓	📌	Research into competitors	2 days	Mon 15/11/10	Tue 16/11/10
38	✓	📌	Research system architecture	2 days	Thu 18/11/10	Fri 19/11/10
39	✓	📌	Design database schema	6 days	Mon 22/11/10	Mon 29/11/10
40	✓	📌	Interim report submission	0 days	Mon 06/12/10	Mon 06/12/10
41	✓	📌	Implementation	43 days	Thu 13/01/11	Sat 12/03/11
42	✓	📌	White Box Testing	43 days	Thu 13/01/11	Sat 12/03/11
43	✓	📌	Black Box Testing	43 days	Thu 13/01/11	Sat 12/03/11
44	✓	📌	Twitter Extractor: Implementation	13 days	Thu 13/01/11	Mon 31/01/11
45	✓	📌	Re-Evaluation of SA Engine Designs	7 days	Thu 03/02/11	Fri 11/02/11
46	✓	📌	Automated Corpus Generator: Implementation	10 days	Sat 12/02/11	Thu 24/02/11
47	✓	📌	SentiWordNet Wordlist: Implementation	2 days	Thu 24/02/11	Fri 25/02/11
48	✓	📌	WordCount Classifier: Implementation	1 day	Sat 26/02/11	Sat 26/02/11
49	✓	📌	Classifier: Implementation	3 days	Sun 27/02/11	Tue 01/03/11
50	✓	📌	Test Case Creation	2 days	Mon 28/02/11	Tue 01/03/11
51	✓	📌	Poster Creation	6 days	Thu 03/03/11	Thu 10/03/11
52	✓	📌	Naive Bayes Classifier: Implementation	2 days	Sun 06/03/11	Mon 07/03/11
53	✓	📌	Evaluation Tests	4 days	Mon 07/03/11	Thu 10/03/11
54	✓	📌	Website Creation	7 days	Sat 05/03/11	Sun 13/03/11
55	✓	📌	Complete implementation	0 days	Sun 13/03/11	Sun 13/03/11
56	✓	📌	Project poster submission	0 days	Thu 10/03/11	Thu 10/03/11
57	✓	📌	Final report	33 days	Mon 14/03/11	Wed 27/04/11
58		📌	Final report submission	0 days	Thu 28/04/11	Thu 28/04/11
59		📌	Presentation	0 days	Mon 02/05/11	Mon 02/05/11

Appendix H

Please note: a large amount of commit notes are missing purely because of forgetting to. All dates tie in with the logbook.

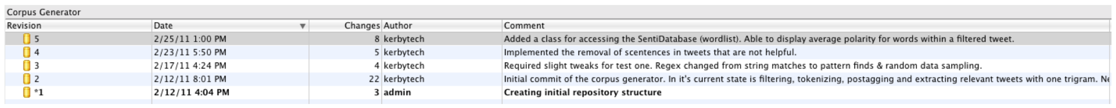
Classifier



Classifier Custom

Revision	Date	Changes	Author	Comment
5	3/7/11 10:43 AM	12	kerbytech	Created a new Results class (universal for all classifiers) so that results can easily be calculated.
4	3/1/11 7:40 PM	10	kerbytech	Improvements: Utilising new emotional wordlist & tweets are read in from the test case inside a DB. Now is able automatically assess accuracy.
3	2/28/11 12:44 PM	23	kerbytech	Basics are complete. Still requires a lot of work, see subtitle "progress" in logbook (todays date).
2	2/27/11 10:58 AM	5	kerbytech	Initial commit for the Custom Classifier. ...
***1**	**2/27/11 10:56 AM**	**3**	**admin**	**Creating initial repository structure**

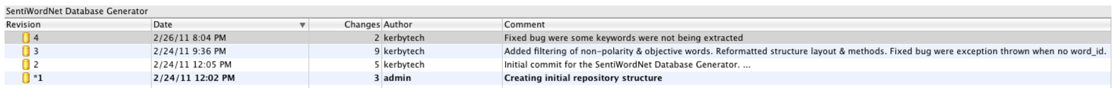
Corpus Generator



Corpus Generator

Revision	Date	Changes	Author	Comment
5	2/25/11 1:00 PM	8	kerbytech	Added a class for accessing the SentiDatabase (wordlist). Able to display average polarity for words within a filtered tweet.
4	2/23/11 5:50 PM	5	kerbytech	Implemented the removal of scentences in tweets that are not helpful.
3	2/17/11 4:24 PM	4	kerbytech	Required slight tweaks for test one. Regex changed from string matches to pattern finds & random data sampling.
2	2/12/11 8:01 PM	22	kerbytech	Initial commit of the corpus generator. In it's current state is filtering, tokenizing, postagging and extracting relevant tweets with one trigram. Ne
***1**	**2/12/11 4:04 PM**	**3**	**admin**	**Creating initial repository structure**

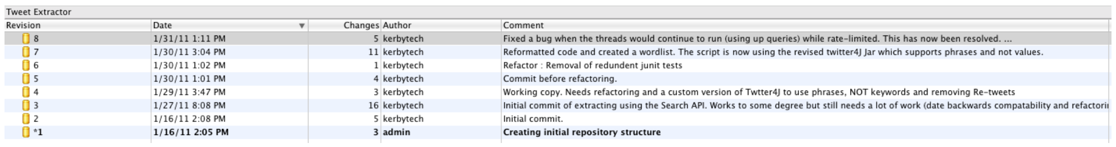
SentiWordNet Wordlist Generator



SentiWordNet Database Generator

Revision	Date	Changes	Author	Comment
4	2/26/11 8:04 PM	2	kerbytech	Fixed bug were some keywords were not being extracted
3	2/24/11 9:36 PM	9	kerbytech	Added filtering of non-polarity & objective words. Reformatted structure layout & methods. Fixed bug were exception thrown when no word_id.
2	2/24/11 12:05 PM	5	kerbytech	Initial commit for the SentiWordNet Database Generator. ...
***1**	**2/24/11 12:02 PM**	**3**	**admin**	**Creating initial repository structure**

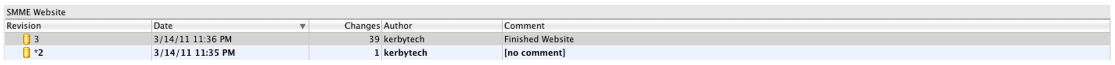
Twitter Extractor



Tweet Extractor

Revision	Date	Changes	Author	Comment
8	1/31/11 1:11 PM	5	kerbytech	Fixed a bug when the threads would continue to run (using up queries) while rate-limited. This has now been resolved. ...
7	1/30/11 3:04 PM	11	kerbytech	Reformatted code and created a wordlist. The script is now using the revised twitter4J Jar which supports phrases and not values.
6	1/30/11 1:02 PM	1	kerbytech	Refactor : Removal of redundent junit tests
5	1/30/11 1:01 PM	4	kerbytech	Commit before refactoring.
4	1/29/11 3:47 PM	3	kerbytech	Working copy. Needs refactoring and a custom version of Twtter4J to use phrases, NOT keywords and removing Re-tweets
3	1/27/11 8:08 PM	16	kerbytech	Initial commit of extracting using the Search API. Works to some degree but still needs a lot of work (date backwards compatability and refactori
2	1/16/11 2:08 PM	5	kerbytech	Initial commit.
***1**	**1/16/11 2:05 PM**	**3**	**admin**	**Creating initial repository structure**

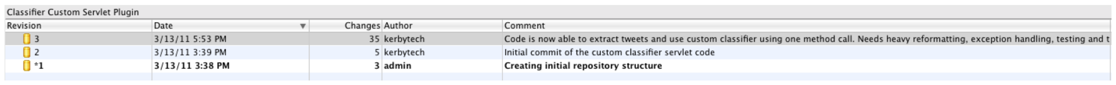
Website



SMME Website

Revision	Date	Changes	Author	Comment
3	3/14/11 11:36 PM	39	kerbytech	Finished Website
***2**	**3/14/11 11:35 PM**	**1**	**kerbytech**	**[no comment]**

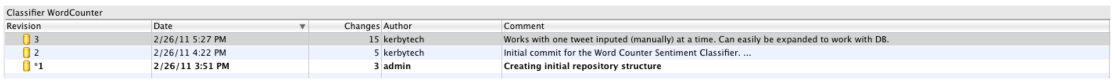
SMME Classifier and Extractor Servlet Plugin



Classifier Custom Servlet Plugin

Revision	Date	Changes	Author	Comment
3	3/13/11 5:53 PM	35	kerbytech	Code is now able to extract tweets and use custom classifier using one method call. Needs heavy reformatting, exception handling, testing and t
2	3/13/11 3:39 PM	5	kerbytech	Initial commit of the custom classifier servlet code
***1**	**3/13/11 3:38 PM**	**3**	**admin**	**Creating initial repository structure**

Word-Count Classifier



Classifier WordCounter

Revision	Date	Changes	Author	Comment
3	2/26/11 5:27 PM	15	kerbytech	Works with one tweet inputed (manually) at a time. Can easily be expanded to work with DB.
2	2/26/11 4:22 PM	5	kerbytech	Initial commit for the Word Counter Sentiment Classifier. ...
***1**	**2/26/11 3:51 PM**	**3**	**admin**	**Creating initial repository structure**