

**Project Report: Sentiment Analysis of Twitter data**

**Himansu Badhai  
Shrey Shrivastava  
Subramani Ramkumar  
Swapnil Balakrishna**

## Table of Contents

Problem Statement.....	3
Approach.....	3
Objectives.....	3
Background.....	3
Motivation.....	4
Challenges.....	4
Data Description.....	4
Pre-processing the data.....	5
Building the training model.....	6
Predicting the sentiment.....	6
Methods.....	6
-LingPipe Sentiment Analysis.....	6
Results.....	7
Appendix.....	12

## Problem Statement

The problem in sentiment analysis is to classify the polarity of a given tweet. The goal is to classify a given tweet as positive or negative depending on the sentiment that it conveys.

## Approach

- Data Gathering – Data consumed using the Twitter API was analysed to see if it was adequate.
- Understanding the data- Understood the data and scrutinized it for identifying key components which were influential for performing analysis.
- Extracting the data needed for analysis- Data preparation was done for analysis and results.
- Data Transformation- Stopwords/Url/Duplicate tweets removal, Stemming and POS tagging.
- Analysing results- Classification using Naïve Bayes and Logistic Regression, Computing Accuracy of the model
- Impact-The impact of recommendations if incorporated by the client was simulated.

## Objectives

- Train a model to identify the sentiment of a given tweet as positive or negative
- Use the trained model to predict the sentiment of any given tweet.

## Background

Twitter has become a leading social media platform owing to its unique microblogging service where registered users can share messages with the world. (These messages are known as tweets). These tweets can be up to 140 characters and can express any thoughts or feelings of a given individual.

The objective of this project is to build an application that accurately classify a given tweet as positive or negative based on it's sentiment. Our hypothesis is that we can predict the sentiment of a given tweet with a greater degree of accuracy by building a thorough training model using Machine Learning techniques.

For the purpose of this project, we identified sentiment conveyed in a given tweet as positive or negative. Shown below are a couple of examples

<b>Tweet</b>	<b>Sentiment</b>
Avenk3: Go Blackhawks go!!	Positive
Mscully: Finals Week! Ugh! :/	Negative

## Motivation

It is well known today that lot of people take to Twitter each day to share their opinions with the world. There are close to 232 million registered users on Twitter and close to 500 million tweets being shared every day. This is a lot of data and most of the data is informal text since it is people talking about their thoughts or feelings. It can contain lot of symbols, emoticons, and incorrect usage of grammar. Given the large set of data and the complexity of the type of data, we felt this was a challenging project to take on and summarize the overall sentiment of a given tweet.

## Challenges

Tweets are limited to 140 characters and hence very short. It can be something as simple as an acronym or a headline. The language used can be very informal or with incorrect grammar, repeated words. To mine through these kind of text and get an understanding of the underlying sentiment was a challenging task.

## Data Description

Although a single tweet contains a maximum of only 140 characters, it can contain characters of a variety of different data types.

The table below lists the different types of datum that can be used in a tweet.

Type	Example
Text	Free Donuts tonight at Dunkin Donuts!
Numbers	50 feared dead at a plane crash in Ukraine
Acronmys	Going to Africa on a safari tonight! YOLO
Negation words	No one showed up at the game yesterday
URL	<a href="http://www.patagonia.com">www.patagonia.com</a> sale going on now! Get it before it's all gone!
User names	@PeterV can you feel the cold coming?
Hash tags	1-0 to the Gunners #GoGunnarsGo
Special symbols	This day is going bad! #@!#@

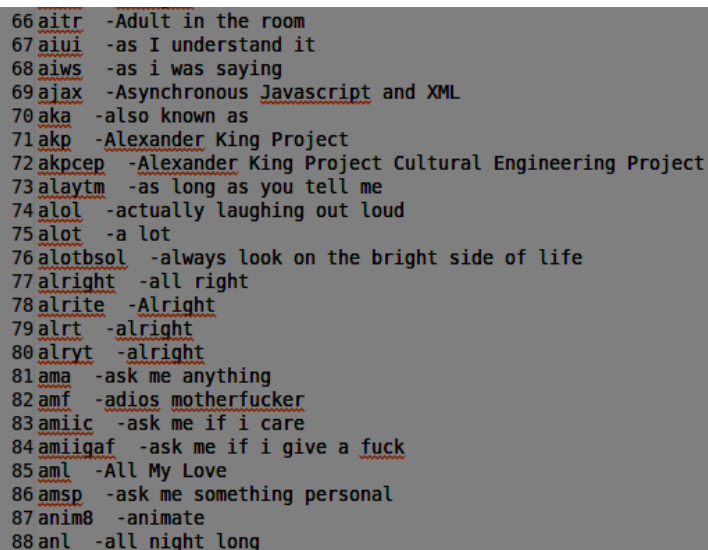
Although a single tweet contains a maximum of only 140 characters, it can contain characters of a variety of different data types. Using the model that we developed we cleansed the data to remove all auxiliary words in a given tweet.

## Pre-processing the data

We process all the tweets as follows:

1. Replace all the URL's with blank.
2. Replace all hashtags (#) and targets (For ex: @James) with blank.
3. Replace acronyms and emoticons with their assigned meanings.
4. Replace negations with word 'NOT'.
5. Tagged the tweets.
6. Stemmed the words to its root word.

For replacing acronyms we have prepared an acronym dictionary from all the slangs listed in [www.noslang.com/dictionary/](http://www.noslang.com/dictionary/). The dictionary has translations for 5300 acronyms. Our program compares replaces the acronyms with their translated meanings in the tweet.



```
66 aitr -Adult in the room
67 aiui -as I understand it
68 aiws -as i was saying
69 ajax -Asynchronous Javascript and XML
70 aka -also known as
71 akp -Alexander King Project
72 akpcep -Alexander King Project Cultural Engineering Project
73 alaytm -as long as you tell me
74 alol -actually laughing out loud
75 alot -a lot
76 alotbsol -always look on the bright side of life
77 alright -all right
78 alrite -Alright
79 alrt -alright
80 alryt -alright
81 ama -ask me anything
82 amf -adios motherfucker
83 amiic -ask me if i care
84 amiigaf -ask me if i give a fuck
85 aml -All My Love
86 amsp -ask me something personal
87 anim8 -animate
88 anl -all night long
```

Below image is the snapshot of our acronyms dictionary.

For handling the presence of emoticons we have compiled a list of 160 emoticons listed in Wikipedia ([http://en.wikipedia.org/wiki/List\\_of\\_emoticons](http://en.wikipedia.org/wiki/List_of_emoticons)) and have assigned them polarity according to their usage. For ex: “:)” this smile symbol has been assigned a positive polarity as it is mostly used for expressing happiness.

We have also replaced negations with 'NOT' and for selecting negations we have referred to the negative words provided by the Grammarly handbook ([www.grammarly.com/handbook/sentences/negatives/](http://www.grammarly.com/handbook/sentences/negatives/)). Our algorithm converts can't to can NOT (Since, n't is the negation here) and cannot to NOT.

We have used Stanford's POS tagger to tag the tweets and referred to Apache Lucence Stemming algorithm to stem the words. In any text adjectives are good indicators of subjective, evaluative sentences. However an adjective might give us an understanding of subjectivity but an adjective alone could not shed much light on a sentence semantic orientation. For ex: the adjective “unpredictable” might have a negative orientation in automotive context (unpredictable steering) but it might have a positive context in a movie review(unpredictable

plot). Thus, we have implemented an algorithm which extracts the two consecutive words in a tweet if and only if the combination of the two extracted words satisfies any of the below mentioned conditions:

First Word	Second Word	Third Word(not selected)
JJ	NN or NNS	anything
RB, RBR, RBS	JJ	Not NN nor NNS
JJ	JJ	Not NN nor NNS
NN or NNS	JJ	Not NN nor NNS
RB, RBR or RBS	VB, VBD, VBN or VBG	anything

For ex: if we take the third case then the combination JJ , JJ will only be selected if the third consecutive word is not a category of noun(NN or NNS).

After cleaning the tweets by applying all the above mentioned steps we apply the input to train our models.

### **Building the training model**

For training the model we have used two classification algorithms namely Logistic Regression and Naïve Bayes. The details for each of these classification algorithms are explained in the “Methods” section of the document.

### **Predicting the sentiment**

Based on the classification algorithm selected each tweet that is entered by the user is associated with one of the two categories namely “Positive” or “Negative”.

## **Methods**

### **-LingPipe Sentiment Analysis**

LingPipe’s Language Classification Framework internally uses logistic regression model to predict whether a tweet is positive or Negative. To perform sentiment analysis LingPipe’s API focuses on two major classification approaches:

1. Classify subjective (opinion) vs. objective (fact) tweets
2. Classify positive vs. negative tweets

Our program reads an input file that contains the sentiment and the tweet and then puts them to a training directory under sub folders Positive and Negative based on the tweet. Each tweet is stored in separate text file. This training directory is used to train the model and stores it to a file known as classifier.txt

The accuracy of this model is shown in the table on the following page. The steps required to execute LingPipe Sentiment Analysis are mentioned in the Appendix.

Here is a screenshot of how the basic polarity classification is done by LingPipe

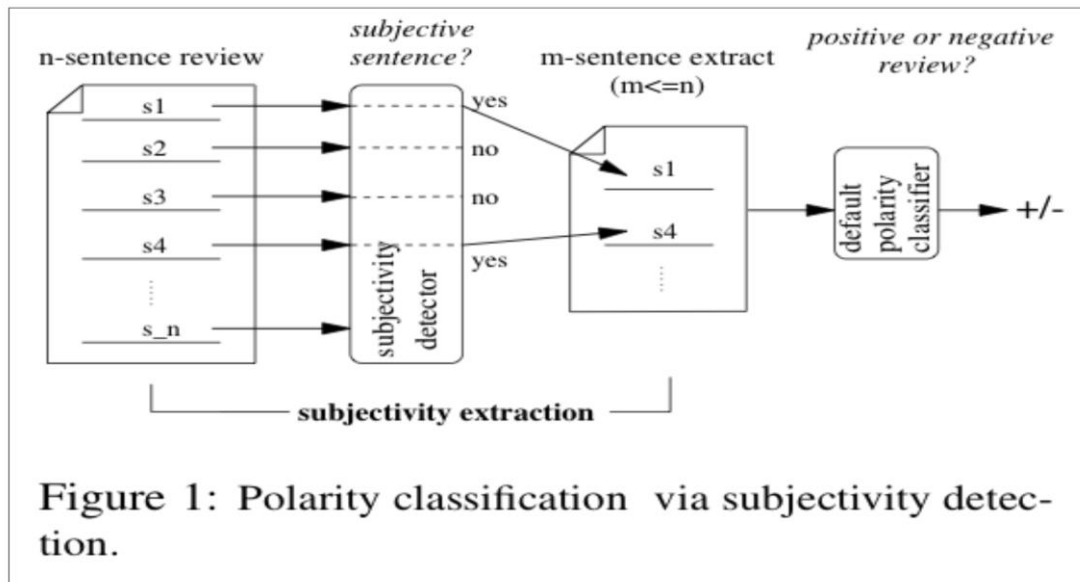


Figure 1: Polarity classification via subjectivity detection.

#### -Accuracy for Logistic Regression Model

Gram Level	Accuracy Details
Unigram (Gram =1)	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1793 % Correct=63.20056397603102
Bigram (Gram =2)	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1818 % Correct=64.08177652449771
Gram = 3	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1679 % Correct=59.18223475502291
Gram = 4	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1657 % Correct=58.40676771237222
Gram = 5	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1633 % Correct=57.560803665844205
Gram = 6	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1641



	% Correct=57.842791681353546
Gram = 7	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1641 % Correct=57.842791681353546
Gram = 8	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1633 % Correct=57.560803665844205
Gram = 12	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1630 % Correct=57.455058160028194
Gram = 20	Training Cases=4693 Number of Test Cases=2837 Number of Correct=1589 % Correct=56.009869580542826

### **-Naïve Bayes Classification Using Mahout Distribution**

A Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model". In simple terms, a Naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes classifier considers all of these properties to independently contribute to the probability.

In our project we are using the Mahout distribution to classify tweets using the Naïve Bayes classification algorithm. The algorithm works by using a training set which is a single file with a single tweet on each line that is associated to a category ("0-Negative" and "1-Positive"). Using this set, the classifier determines for each word, the probability that it makes a tweet to belong to each of the considered categories. To compute the probability that a tweet belongs to a category, it multiplies together the individual probability of each of its word in this category. The category with the highest probability is the one the tweet is most likely to belong to.

The High-level algorithm states for each word in each line we do the following two things:

- 1) Calculate "Weighted-Normalized TF-IDF" which is given as the TF-IDF calculated as a standard IDF multiplied by the weight normalized TF.

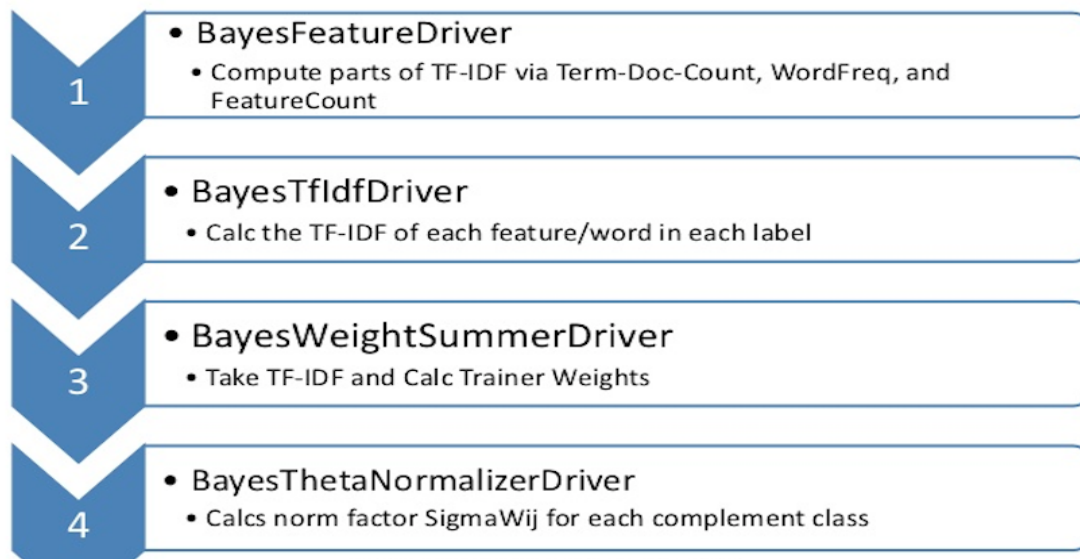
- 2) We calculate the sum of W-N-Tf-idf for all the features in a label called Sigma<sub>k</sub>, and alpha<sub>i</sub> equal 1.0

$$Weight = \text{Log} [(W\_N\_Tf\_idf + \alpha\_i) / (Sigma\_k + N)]$$

The diagram on the next page describes the workflow followed by Mahout to train the Naïve Bayes Classification Model .

## BayesDriver Training Workflow

Naïve Bayes Training MapReduce Workflow in Mahout



### -Issues with Naïve Bayes Classification:

#### 1) Violation of Independence Assumption:

Naïve Bayesian Classifiers assume that the effect of an attribute is independent of the values of other attributes. This assumption is called class conditional independence. It is made to simplify the computations involved, and in that sense, called "Naive". To overcome this issue we can make use of "Bayes Belief Networks".

#### 2) Zero Conditional Probability Problem:

If a given class and a feature value never occur together in a training set then the frequency based estimate probability will be zero. This is problematic because it will wipe out all the probabilities when they are multiplied together. To eliminate zeroes we add one or add Laplace smoothing which simply adds one to each count.

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

## Results

In our program we are using approximately 3200 manually labelled tweets to train our model. The accuracy of this model is 55-75% based on the gram level. Shown below are examples how the application works and the output it produces. Attached screenshot below shows output for of a Negative tweet and a Positive tweet input given by user.



### Twitter Sentiment Calculator

**Please Enter your Tweet**

Tweet:

Classification Method: ☐ Naive Bayes ☒ Logistic Regression

**Sentiment Results**

Tweet: my iphone broke in just 2 days

Result: Negative

### Negative Tweet



## Twitter Sentiment Calculator

### Please Enter your Tweet

Tweet:

what a great opening by Lebron James at the NBA Conference

Classification Method:



Naive Bayes



Logistic Regression

submit

### Sentiment Results

Tweet:

what a great opening by  
Lebron James at the  
NBA Conference

Result:

Positive

**Positive Tweet**

## Appendix

### Read Me for pre-processing the tweets

#### Step I

Import the TwitterProject in eclipse and copy the file with all the tweets in the folder named Files.

In this case the input file is named InputTweetsWithSentiments.

Make sure to change the input path in the driver class.

Variable `String inputPath` has the inputPath and `String outputPath` has the outputpath.

#### Step II

Run the DriverClass as java application and the output will be generated in the project folder by the name of RefinedTweets.

Output generated by the mapper reducer will be in the following format:

Tweet	happy man people	1
Key	Refined tweet	Sentiment score

[Click here](#) for the explanation of map reduce program.

#### Step III

Import ExtractTweets project and copy the part-00000 file inside the project folder.

#### Step IV

Now execute ExtractTweets.java as java application and the output will be generated in the Output folder in the file named InputTweets.

[Click here](#) for the explanation of ExtractTweets.java class

#### Step V

Import the Taggers project in eclipse and copy the InputTweets generated by ExtractTweets.java class in the Taggers project.

## Step VI

Execute the MainClass as java application and two output files will be generated in the File folder with **names:AfterPOSTweets** and **TweetsAfterStemming**.

**AfterPOSTweets** text file is the output of **TaggingClass.java** and **TweetsAfterStemming** text file is the output of **Stemmer.java**

[Click here](#) for the explanation of Taggers project.

## Step VII

Copy the TweetsAfterStemming text file in the ExtractTweets project and execute the CombineTweetsAndSentiment class as java application. The execution of program will generate a text file named FinalOutput in the output folder which will contain the ouput in the following format:

0	disappoint feel
Sentiment Score	Final refined tweet

This FinalOutput text file will be used to train the models.

[Click here](#) for the explanation of CombineTweetsAndSentiment java class

## Explanation of programs:

The com.tweet.cleaner package in TwitterProject is responsible for cleaning the tweets. The map reduce implementation of this program follows the following steps to clean the data:

1. Replace the URL's and remove the hashtags from the tweets.
2. Replace the emoticons with their assigned meaning.
3. Replace the acronyms (slangs used) with their meanings.
4. Replace all the negations word with word NOT.
5. Remove the stop words.

### TweetCleanerMapper.java

The mapper class implements the logics of all the above mentioned steps. It has list of punctuations, array of positive emoticons, array of negative emoticons and an array of negations (cannot, not etc.) as class variables and two hashmaps which stores the stopwords and list of acronyms.

TweetCleanerMapper class implements the following methods:

1. `readListOfSlangs()`

This method reads the ListOfSlang file and tranfer its contents in the acronymsMap with acronym as key and its full explanation as the value.

## 2. readStopWords()

This method reads the stopWords file and store all the stopwords in the stopWords map.

## 3. replaceURLsatTheRateandHashTagSymbol(String tweet)

This method replaces all the URL's, @'s and # tags with an empty String.

The method uses regular expressions to detect all the URL's, @'s and # tags in the tweets and uses the String's replaceAll method to replace the detected URL's, @'s and # tags with an empty String.

## 4. replaceEmoticons(String tweet)

The method uses array of positiveEmoticons and array of negativeEmoticons to detect the emoticons and then uses String's replace method to replace the emoticon with its assigned meaning.

## 5. replaceAcronyms(String tweet)

The method tokenizes the tweet and search for the acronyms from the acronymsMap and replaces all the acronyms with their meanings.

## 6. replaceNegations(String tweet)

The method replaces all the negation word with a single word NOT.

**NOTE: can't is changed to can NOT and cannot is changed to NOT**

## **TweetCleanerReducer.java**

The reducer class handles duplicate tweets. It implements the logic to remove the duplicate tweets in the input file.

[Go back](#)

## **ExtractTweets.java**

This class extracts the tweets from the tweet-sentiment combination and stores all the tweets in a different file.

[Go Back](#)

## **Taggers project**

Tagger project has following three classes:

1. MainClass
2. Stemmer Class
3. TaggingClass

The MainClass reads every tweet from the file and then calls the Tagging class which returns the POS tagged tweet. MainClass then selects the following POS tag combinations:

First Word	Second Word	Third Word
JJ	NN or NNS	anything
RB, RBR, RBS	JJ	Not NN nor NNS
JJ	JJ	Not NN nor NNS
NN or NNS	JJ	Not NN nor NNS
RB, RBR or RBS	VB, VBD, VBN or VBG	anything

**For second, third and fourth combination code implements the following logic: if the first word is adjective (JJ) , the word after that is also adjective(JJ) and if the third word is noun (NN or NNS) then the combination will not be selected.**

Tagging class uses Stanford POS tagger jar to tag the tweets.

After generating the output file with tagged tweets MainClass calls stemmer class which stems all the words in each tweet to its root word. **For ex:** words will be changed to word.

There will be various empty Strings in the output of Tagger class as many tweets will not comply to any of the above combinations and thus POS tagger would generate few empty Strings.

[Go Back](#)

### **CombineTweetsAndSentiment.java**

This class combines the stemmed tweets and sentiments of each line, if the line is empty then that tweet is not considered.

[Go Back](#)

### **Read Me (For running the WebApplication)**

1. Download the war file (TwitterSentimentAnalysis.war) from the assignment folder.
2. Import the project into your eclipse workspace.
3. Run the project on Tomcat server.
4. You should be able to see the home page of the application now.
5. Enter the tweet you want to predict the sentiment for.
6. Choose the classification method and submit.
7. Your results will be displayed on the box on the same page.

To check the accuracy of the model, we have written a standalone java program. Follow the below steps to calculate the accuracy.

1. Go to CalculateAccuracy.java class under the package edu.uic.ids594.accuracy.
2. Run the java class. You can see the output in the console.

### **Notes:**

When you run the application in Mac, classifier.txt and the training directory is created under your eclipse content files. To view these files, follow the following steps-

1. Go to your eclipse installation directory.
2. Right click on the eclipse.exe icon and click on Show package contents.
3. Both the training directory and the classifier file are generated under  
Contents → MacOS



Once the classifier is created, it can be used to calculate the sentiment for any tweet given by the user. The TweetSetimentClassifier.java class reads the classifier file generated in the previous step and classifies the tweet as positive or negative.

For **Naïve Bayes**, the model is created in the in the “output” folder under same directory as mentioned above.

Once the model is trained, you don't need to run the train the model again. Training the model takes some time because of the huge input dataset. Comment the call to trainMymodel method in LogisticRegressionSentimentCalculator.java class and trainNaiveBayes method in NaiveBayesSentimentCalculator.java class for subsequent execution of the program. This improves the performance of the application.

**References:**

List of Slangs: [www.noslang.com/dictionary/](http://www.noslang.com/dictionary/)

List of emoticons: [http://en.wikipedia.org/wiki/List\\_of\\_emoticons](http://en.wikipedia.org/wiki/List_of_emoticons)

Negations: [www.grammarly.com/handbook/sentences/negatives/](http://www.grammarly.com/handbook/sentences/negatives/)

Study of Semantic Orientation by National Research Council Canada

Research on Sentiment Analysis of Twitter data by Columbia University