Romain Bui - 80975422

**Big Data Case Study in Social Media**

**“Smile and Tweet – The usage of Smileys to train a Sentiment Analysis on tweets”**

**Natural Language Processing (NLP):** A filed in computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) language which uses different algorithms to process linguistic information.[[1]](#footnote-1)

**Sentiment Analysis:** A topic of natural language processing that consists in extracting information from text materials.[[2]](#footnote-2)

**Project Description:** I used Mahout to implement a Naïve Bayes Classification Algorithm on tweets using a training set automatically created from smileys in the messages. The idea consists in testing if such an approach in creating an automatic training set is enough to have good results on a classical algorithm.

**Problem:** The amount of “qualitative” information is growing exponentially with the social networks. Sentiment helps the data scientist to extract valuable information from this source. As pointed out during the class the training process is an important part of the process. Only the big companies can afford to manually create these training sets. The international nature and meaning of the smileys as well as their omnipresence on the web could be a tool to automatically create this set. We propose to use such a training set and test its strength.

**Objective:** Demonstrate or conceal the strength of such a training set in the context of a Naïve Bayes Algorithm.

**Challenge:** The training set and the test set are in different format, since the training set was automatically created we could use this set in both the training and testing format. The formats consist of a text file for the training set and a text file for the testing set. We had to develop custom java function in order to create sequence type of object that can then be properly used using Mahout.

**DataSet:** The data consists of :

- The training Corpus from Sentiment 140[[3]](#footnote-3) (Stanford) consisting of 1 600 000 tweets qualified as positive or negative using the smileys to determine their orientation.

- The testing Corpus from ravikiranj on github[[4]](#footnote-4) consisting of 5406 tweets properly classified by sentiment.

**Technology** : Java 7.0, Mahout 0.9.

**NOTA BENE:** The codes and data are in the zip file in the source folder.

**Smiley and Training Set:**

In this section we explain the methodology used by help sentiment 140 to create the training set. Alec Go, Richa Bhayani and Lei Huang from Stanford explain in their paper “Twitter Sentiment Classification using Distant Supervision”[[5]](#footnote-5) how they constructed the training set that we use in our study. The classification in first and foremost done through the emoticons following:

|  |  |
| --- | --- |
| Positive | Negative |
| :) | :( |
| :-) | :-( |
| : ) | : ( |
| :D |  |
| =) |  |

All the other tweets were removed. Once the classification is done all the emoticons are removed in order to avoid the training process to highly focus on them.

**Steps:**

1. **INSTALLATION OF MAHOUT**

The first step consists in installing Mahout, this has already been done for previous homework but we take the time here to quickly re-explain the steps. The idea consists in installing mahout and modifying the bash\_profile to add the proper external variables. The code is as followed:

sudo yum install mahout

vi .bash\_profile

MAVEN\_HOME=/usr/lib/apache-maven-3.2.1

export MAVEN\_HOME

MAHOUT\_HOME=/usr/lib/mahout

export MAHOUT\_HOME

HADOOP\_HOME=/usr/lib/hadoop

export HADOOP\_HOME

HADOOP\_CONF\_DIR = /usr/lib/hadoop-0.20-mapreduce/conf

export HADOOP\_CONF\_DIR

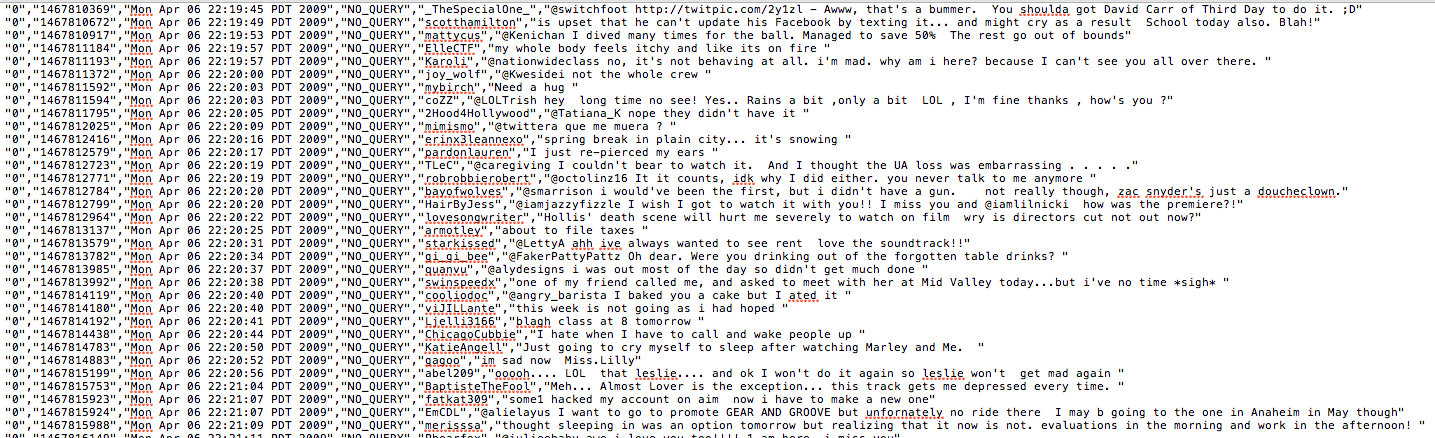
PATH=$JAVA\_HOME/bin:$PATH:$HOME/bin:$MAVEN\_HOME/bin:$MAHOUT\_HOME

export PATH

1. **TRAINING SET INTO SEQUENCE FILE**

This step differs from the usual approach we took during class. As a matter of fact in the previous homework we used the command ***seqdirectory*** from mahout in order to transfer the files into Haddop Sequence Files. However our training set is a csv (train.csv) of the form:

Sentiment, ID, Date, QueryIfAny, Tweet



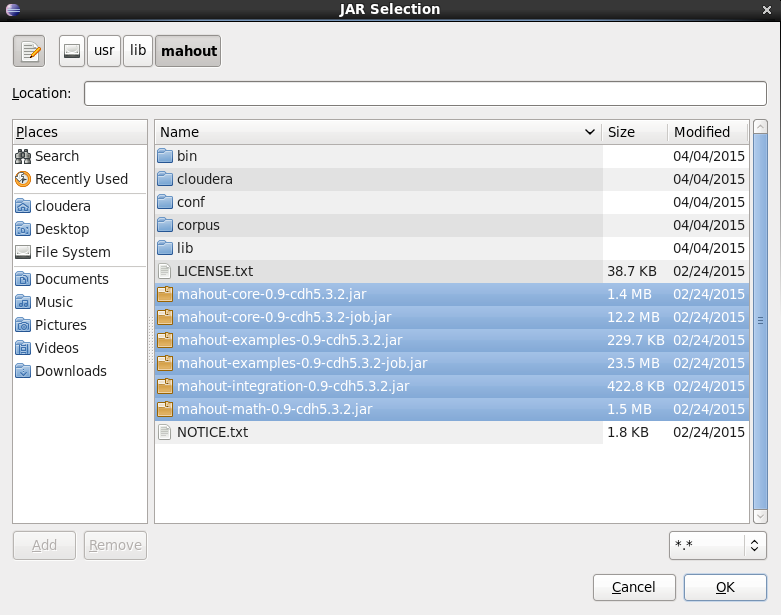
Sentiment is defined as 0:negative, 4:positive. Obviously one could write a script in a given language and transform this csv file into several files properly ordered into folders. We believe that this would be non-sustainable fix. Given that data in such format is highly probable, we will show in the next pages how we handle the csv, and how we created the sequence file using a custom java class. Without further details here is the code we will discuss (TweetToSeq.java).

1. **package** Mahout**;**
2. **import** java**.**io**.**BufferedReader**;**
3. **import** java**.**io**.**FileReader**;**
4. **import** org**.**apache**.**hadoop**.**conf**.**Configuration**;**
5. **import** org**.**apache**.**hadoop**.**fs**.**FileSystem**;**
6. **import** org**.**apache**.**hadoop**.**fs**.**Path**;**
7. **import** org**.**apache**.**hadoop**.**io**.**SequenceFile**;**
8. **import** org**.**apache**.**hadoop**.**io**.**SequenceFile**.**Writer**;**
9. **import** org**.**apache**.**hadoop**.**io**.**Text**;**
10. **public** **class** TweetToSeq **{**
12. **public** **static** **void** main**(**String args**[])** **throws** Exception **{**
13. **if** **(**args**.**length **!=** **2)** **{**
14. System**.**err**.**println**(**"Arguments: [input csv file] [output sequence file]"**);**
15. **return;**
16. **}**
17. String inputFileName **=** args**[0];**
18. String outputDirName **=** args**[1];**
20. Configuration configuration **=** **new** Configuration**();**
21. FileSystem fs **=** FileSystem**.**get**(**configuration**);**
22. Writer writer **=** SequenceFile**.**createWriter**(**fs**,** configuration**,** **new** Path**(**outputDirName**),**Text**.class,** Text**.class);**
24. **int** count **=** **0;**
25. BufferedReader reader **=** **new** BufferedReader**(new** FileReader**(**inputFileName**));**
26. Text key **=** **new** Text**();**
27. Text value **=** **new** Text**();**
28. **while(true)** **{**
29. String line **=** reader**.**readLine**();**
30. **if** **(**line **==** **null)** **{**
31. **break;**
32. **}**
33. line **=** line**.**replace**(**"**\"**"**,** ""**);**
35. String**[]** tokens **=** line**.**split**(**","**,** **5);**
36. **if** **(**tokens**.**length **!=** **5)** **{**
37. System**.**out**.**println**(**"Skip line: " **+** line**);**
38. **continue;**
39. **}**

42. String category **=** **new** String**();**
44. **if(**Double**.**parseDouble**(**tokens**[0])** **==** **0)**
45. **{**
46. category **=** "Negative :-("**;**
47. **}**
48. **if(**Double**.**parseDouble**(**tokens**[0])** **==** **4)**
49. **{**
50. category **=** "Positive :-)"**;**
51. **}**
53. //category = tokens[0];
55. String id **=** tokens**[1];**
56. String message **=** tokens**[4];**
57. key**.**set**(**"/" **+** category **+** "/" **+** id**);**
58. value**.**set**(**message**);**
59. writer**.**append**(**key**,** value**);**
60. count**++;**
61. **}**
62. writer**.**close**();**
63. System**.**out**.**println**(**"Wrote " **+** count **+** " entries."**);**
64. **}**
65. **}**

The first lines up until (line 11), are pretty obvious they consists in loading the proper libraries and package. Note here that if you have to properly add the external jars in order for the import to work. The necessary jars are as follow :

Mahout Jar under : Usr/Lib/Mahout



Haddop Jar :

Usr/lib/hadoop

Usr/lib/hadoop/client-0.20

From l15 to l21, we simply create the class TweetToSeq, and start the main. This class contains simply a main that will perform all the necessary operation to create the Sequence File. One could easily notice that there are two arguments to this class: the input csv file and the output sequence file.

From l22 to 26 we instantiate the configuration and store the arguments.

L27, we create a writer which is the tool that will allow us to create the sequence file. On line 30, we create a reader that will allow us to go through the text file line by line. On line 29 we instantiate a counter that allow us to count the number of lines that are treated.

On line 33, we start the loop, which consists on an infinite loop while(true) that will be broken at a stage when a condition is met. This loop goes through the lines. On line 34 we read the line, then if the line is empty (l35), we are at the end of the file so we break the loop. On line 38 we change the quotes for nothing in order to avoid calibrating our model on this. We then break the line into tokens, in other words we use the coma to define the tokens and extract the Sentiment, ID, Date, QueryIfAny, Tweet into an array (l39).

If for some reason the line is not valid we discard it (l41). We create the variable “category” which holds the category/sentiment of the line we are currently looking at. We use the first token (l49 – 56) to determine if the line is in the category Negative or Positive. We then define the variable ID (l60) as being the ID of the tweet, and the variable message (l61) to store the tweet.

Finally on l64, we append the writer with a Key Value with the key being : /Category/ID and the Value being the Message.

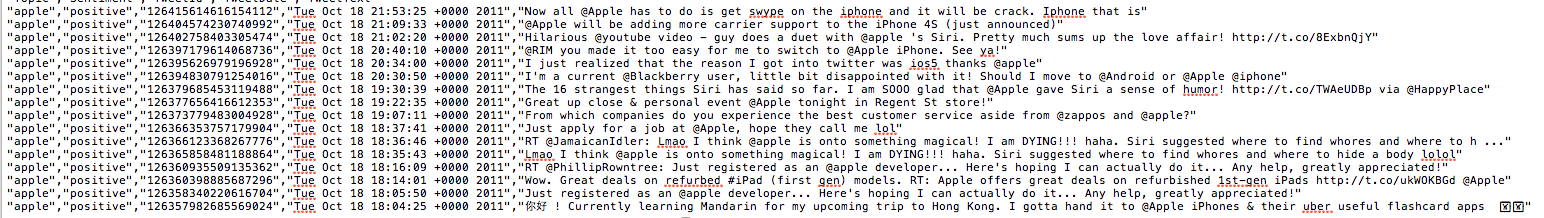
Once the loop is over, a.k.a we are at the end of the file (l67) we close the writer.

1. **TESTING SET INTO SEQUENCE FILE**

Since we want to test the strength of a methodology consisting in automatically building the training set using the smileys, we cannot use the same sample to test the strength of the algorithm. We choose another corpus from github consisting of international tweets towards apple.

The file is a text file which is constructed as follow:

Topic, Sentiment, TweetID, TweetDate, TweetText



As one can easily guess, we use a similar approach, a.k.a a java class, in order to process the text file into a Hadoop sequence file. Using a custom class offers high flexibility and allows us to properly treat this file even though it has a different structure compare to the previous one. Without further details, here is the code we will explain (TweetToSeq\_test.java):

1. **package** Mahout**;**
2. **import** java**.**io**.**BufferedReader**;**
3. **import** java**.**io**.**FileReader**;**
4. **import** org**.**apache**.**hadoop**.**conf**.**Configuration**;**
5. **import** org**.**apache**.**hadoop**.**fs**.**FileSystem**;**
6. **import** org**.**apache**.**hadoop**.**fs**.**Path**;**
7. **import** org**.**apache**.**hadoop**.**io**.**SequenceFile**;**
8. **import** org**.**apache**.**hadoop**.**io**.**Text**;**
9. **import** org**.**apache**.**hadoop**.**io**.**SequenceFile**.**Writer**;**
10. **public** **class** TweetToSeq\_test **{**
12. **public** **static** **void** main**(**String args**[])** **throws** Exception **{**
13. **if** **(**args**.**length **!=** **2)** **{**
14. System**.**err**.**println**(**"Arguments: [input csv file] [output sequence file]"**);**
15. **return;**
16. **}**
17. String inputFileName **=** args**[0];**
18. String outputDirName **=** args**[1];**
20. Configuration configuration **=** **new** Configuration**();**
21. FileSystem fs **=** FileSystem**.**get**(**configuration**);**
22. Writer writer **=** SequenceFile**.**createWriter**(**fs**,** configuration**,** **new** Path**(**outputDirName**),**Text**.class,** Text**.class);**
24. **int** count **=** **0;**
25. BufferedReader reader **=** **new** BufferedReader**(new** FileReader**(**inputFileName**));**
26. Text key **=** **new** Text**();**
27. Text value **=** **new** Text**();**

30. **while(true)** **{**
31. String line **=** reader**.**readLine**();**
32. **if** **(**line **==** **null)** **{**
33. **break;**
34. **}**



39. line **=** line**.**replace**(**"**\"**"**,** ""**);**
41. String**[]** tokens **=** line**.**split**(**","**,** **5);**
42. **if** **(**tokens**.**length **!=** **5)** **{**
43. System**.**out**.**println**(**"Skip line: " **+** line**);**
44. **continue;**
45. **}**

48. String category **=** **new** String**();**
50. **if(**tokens**[1].**equals**(**"negative"**))**
51. **{**
52. category **=** "Negative :-("**;**
53. **}else** **if(**tokens**[1].**equals**(**"positive"**))**
54. **{**
55. category **=** "Positive :-)"**;**
56. **}else{**
57. System**.**out**.**println**(**"Skip line: " **+** line**);**
58. System**.**out**.**println**(**tokens**[1]);**
59. **continue;**
60. **}**
62. //category = tokens[1];
64. String id **=** tokens**[2];**
65. String message **=** tokens**[4];**
66. key**.**set**(**"/" **+** category **+** "/" **+** id**);**
67. value**.**set**(**message**);**
68. writer**.**append**(**key**,** value**);**
69. count**++;**
70. **}**
71. writer**.**close**();**
72. System**.**out**.**println**(**"Wrote " **+** count **+** " entries."**);**
73. **}**
74. **}**

This script is highly similar to the one presented in the previous section, the main difference lies on the line 53, where this time, we define the category based on the classification presented in the text file (in other words not a 0,4 nomenclature but rather the usage of words “positive” “negative” “neutral”). Note on the l.59 that we exclude all the lines that do not have a classification “positive” or “negative”, in other words we discard the “neutral” lines. We easily see how the usage of custom class offers a highly flexible solution.

1. **COMMAND LINES AND IMPLEMENTATION**

In this section we present the different command lines necessary to implement the Bayes Theorem in Mahout.

We first create the directory in the Hadoop File System using the well-known command “mkdir”:

*hadoop fs –mkdir FinalAssignment*

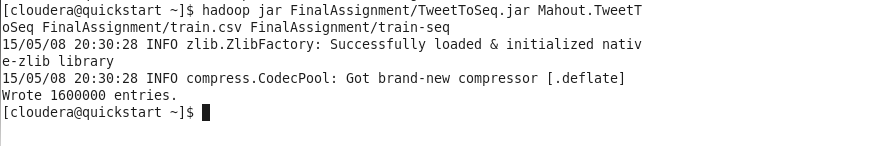
**Sequence File**

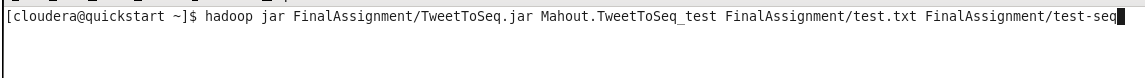
We choose the directory final assignment; one can obviously choose another directory. Once the java classes are properly jared, one can call the *hadoop jar* command to create the sequence file :

*Hadoop jar FinalAssignment/TweetToSeq.jar Mahout.TweetToSeq FinalAssignment/train.csv FinalAssignment/train-seq*

*hadoop jar FinalAssignment/TweetToSeq.jar Mahout.TweetToSeq\_test FinalAssignment/test.txt FinalAssignment/test-seq*

This will create the sequence file under the name train-seq and test-seeq directly into HDFS.



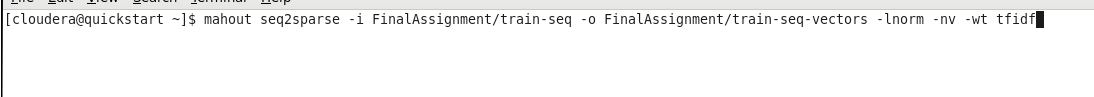


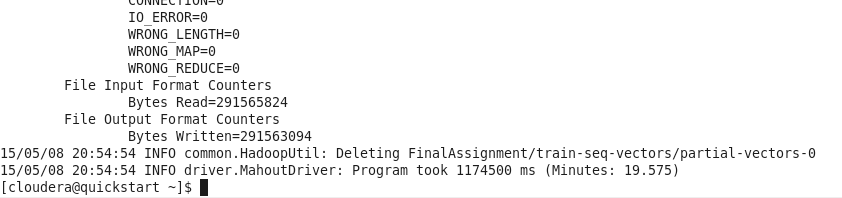


**Train the Model**

Once the sequence files have been properly created one can use the seq2sparse to create the sparse vector in the multi-dimensional vector space of all the terms in the different tweets. The words are transformed in their base form then the weights are calculated. We choose the Euclidian distance with the weight function of type tf-idf (term frequency – inverse document frequency). The idea of tf-idf consists in looking for words that are highly frequent in a specific category of documents while not being frequent in the others. It’s a common practice to identify the key words defining a category.

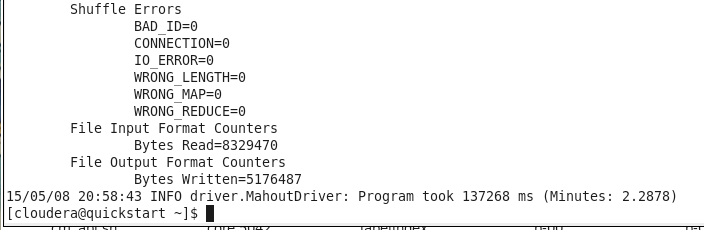
*mahout seq2sparse -i FinalAssignment/train-seq -o FinalAssignment/train-seq-vectors -lnorm -nv -wt tfidf*

**

**

In order to train the model and create the object and create the “model” one can then use the command “trainnb” applied on the tfidf-vectors :

*mahout trainnb -i FinalAssignment/train-seq-vectors/tfidf-vectors -el -o FinalAssignment/model -li labelindex –ow*

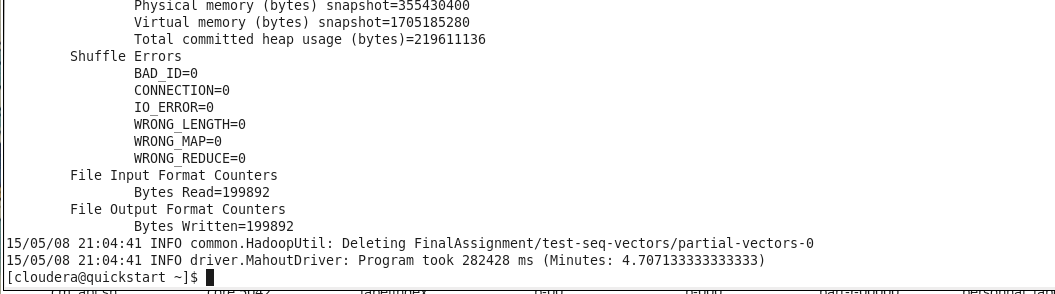
**

**Test the Model**

In order to test the model one must first create the sparse vector as in the train model and then test the model on the test sparse. The sparse command is straight forward and similar to the one in the previous section:

*mahout seq2sparse -i FinalAssignment/test-seq -o FinalAssignment/test-seq-vectors -lnorm -nv -wt tfidf*

**

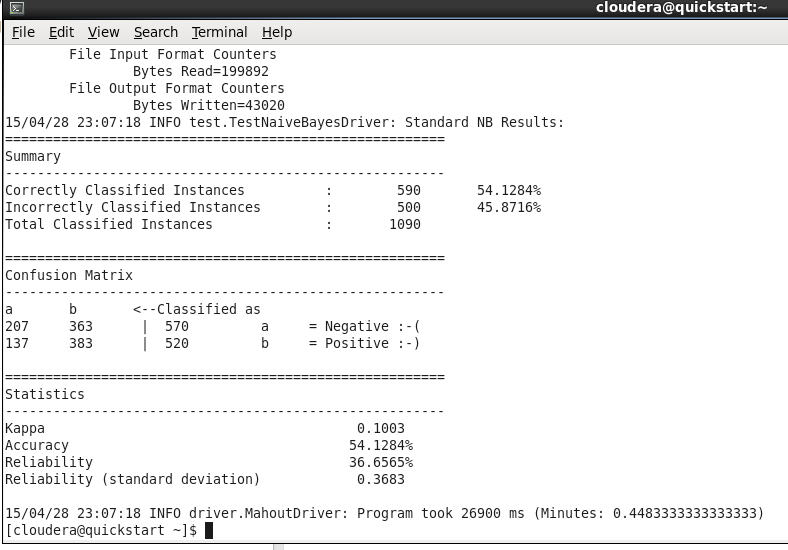
**

Obviously for the sake of comparaison, we must use the same methodology (Euclidean distance combined tf-iddf weighting scheme).

In order to test the model we use the mahout command testnb we use the test vector as input the model created in the previous section as the model and compute the statistics on the model :

*mahout testnb -i FinalAssignment/test-seq-vectors/tfidf-vectors -m model -l labelindex -ow -o FinalAssignment/test-res*

The results obtained are as below:



1. **Discussion**

The results show 54.18% accuracy, which is very low since 50% accuracy could be statistically attributed to pure luck. In this section we will discuss the pro and cons of this methodology and will conclude our report.

Pros.

It is quite obvious that smiley are international, one can then easily get over the barrier language. The approach consisting in using the smiley to create an automatic training is highly appealing since it does not require a manual labeling.

Cons.

There are several downsides to this methodology that most probably explain why the approach was not really successful. First of all, the smiley offers a way to classify into two different states (positive or negative) it is however impossible to have a “neutral smiley”, also the intensity cannot be measure (the number of smiley could be used but this would be highly random and would depend from one individual to another rather than on the sentiment itself.

One obvious pitfall is that the usage of the smiley is used only by a fraction of the population (a.k.a young people mostly); the calibration would then be done only on tweets that have been written by young people. This induce two main issues, first the words tagged to the different sentiment would belong to the vocabulary used by this part of the population which is not representative of the full spectrum of the words. Second, the subjects covered would be very concentrated, therefore using a train model with this train set on set of tweets on politics for example might lead to poor results.

Finally there are several issues that are linked to the sentiment analysis in general: sarcasm and comparisons. Smiley can hardly detect sarcasm, and tweets with several ideas in them would be hardly properly labeled.

Conclusion.

The proposed approach to automatically create the training sample is highly appealing, such an approach has however several room for errors, and the implementation of a classical Naïve Bayes Algorithm showed poor results. On the algorithm side it is important to understand that the Naïve Bayes Algorithm is a very simple one, we probably could have obtained better results using a more sophisticated approach such as support vector machines or neural network. A possible avenue for future research would be to implement these different methodologies and to compare the results.

1. **Link to YouTube Videos**

Summary:

<https://youtu.be/L8SzS0W7y1s>

Full Presentation (2 Parts):

<https://youtu.be/Zgyf5GrbwnM>

<https://youtu.be/K7NxbRmSW-o>

**SOURCES**

Lecture : Predictive Analytics Using Mahout

Sentimen140 (Training Set)

<http://help.sentiment140.com/for-students/>

Full-Corpus.csv (Testing Set)

<https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/data/full-corpus.csv>

Alec Go, Richa Bhayani, Lei Huang, Twitter Sentiment Classification using Distant Supervision

<http://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf>

Frederic Dang Ngoc, Francois Dang Ngoc, “Using the Mahout Naïve Bayes Classifier to automatically classify Twitter messages”

<https://chimpler.wordpress.com/2013/03/13/using-the-mahout-naive-bayes-classifier-to-automatically-classify-twitter-messages/>

1. Definition inspired from Wikipedia [↑](#footnote-ref-1)
2. Idem [↑](#footnote-ref-2)
3. <http://help.sentiment140.com/for-students/> (training.1600000.processed.noemoticons.csv) [↑](#footnote-ref-3)
4. <https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/data/full-corpus.csv> [↑](#footnote-ref-4)
5. <http://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf> [↑](#footnote-ref-5)