# *DETAIL PROJECT REPORT*

***SENTIMENT ANALYSIS***

Table of content

Table of content

1. Introduction

1.1 Context

1.2 Motivations

1.3 Idea

1.4 Sources

2. The Project

2.1 Data

2.2 Resources

2.3 Preprocessing

2.3.1 Emoticons

2.3.2 Case

2.3.3 Targets

2.3.4Acronyms

2.4 Machine Learning

2.4.1 Naive Bayes

2.4.2 Baseline

2.4.3 Improvements

2.4.4 Language Models

3. Conclusion

1. Introduction

**1.1 Context**

This project has been done as a part of my course for the MSc Information Technology

at Hong Kong University of Science and Technology. Supervised by Dr David Rossiter, I had

three months to fulfill the requirements in order to succeed the module. Every three weeks, a

meeting was organized to show and report my progress and fix the next objectives.

**1.2 Motivations**

Being extremely interested in everything having a relation with the Machine Learning,

the independant project was a great occasion to give me the time to learn and confirm my

interest for this field. The fact that we can make estimations, predictions and give the ability for

machines to learn by themselves is both powerful and limitless in term of application

possibilities. We can use Machine Learning in Finance, Medicine, almost everywhere. That’s

why I decided to conduct my project around the Machine Learning.

**1.3 Idea**

This project was motivated by my desire to investigate the sentiment analysis field of

machine learning since it allows to approach natural language processing which is a very hot

topic actually. Following my previous experience where it was about classifying short music

according to their emotion, I applied the same idea with tweets and try to figure out which is

positive or negative.

**1.4 Sources**

Feel free to give me your point of view or ideas for anything you want. I used ipython notebook

which is very useful to understand the entire process of my project since you can follow each

step with the corresponding code. Here is the direct link to it:

https://github.com/12-sg-shalini/Project\_code1/blob/main/Sentiment\_Analysis.ipynb

2. The Project

Sentiment analysis, also refers as opinion mining, is a sub machine learning task where

we want to determine which is the general sentiment of a given document. Using machine

learning techniques and natural language processing we can extract the subjective information

of a document and try to classify it according to its polarity such as positive, neutral or negative.

It is a really useful analysis since we could possibly determine the overall opinion about a selling

objects, or predict stock markets for a given company like, if most people think positive about it,

possibly its stock markets will increase, and so on. Sentiment analysis is actually far from to be

solved since the language is very complex (objectivity/subjectivity, negation, vocabulary,

grammar,...) but it is also why it is very interesting to working on.

In this project I choose to try to classify tweets from Twitter into “positive” or “negative”

sentiment by building a model based on probabilities. Twitter is a microblogging website where

people can share their feelings quickly and spontaneously by sending a tweets limited by 140

characters. You can directly address a tweet to someone by adding the target sign “@” or

participate to a topic by adding an hastag “#” to your tweet. Because of the usage of Twitter, it is

a perfect source of data to determine the current overall opinion about anything.

**2.1 Data**

To gather the data many options are possible. In some previous paper researches, they

built a program to collect automatically a corpus of tweets based on two classes, “positive” and

“negative”, by querying Twitter with two type of emoticons:

● Happy emoticons, such as “:)”, “:P”, “:)”

etc.

● Sad emoticons, such as “:(“, “:’(”, “=(“.

Others make their own dataset of tweets my collecting and annotating them manually which

very long and fastidious.

Additionally to find a way of getting a corpus of tweets, we need to take of having a

balanced data set, meaning we should have an equal number of positive and negative tweets,

but it needs also to be large enough. Indeed, more the data we have, more we can train our

classifier and more the accuracy will be.

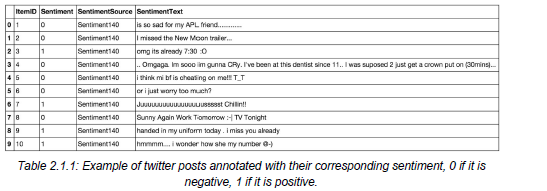
After many researches, I found a dataset of 1578612 tweets in english coming from two

sources: Kaggle and Sentiment140. It is composed of four columns that are *ItemID* , *Sentiment* ,

*SentimentSource* and *SentimentText* . We are only interested by the *Sentiment* column

corresponding to our label class taking a binary value, 0 if the tweet is negative, 1 if the tweet is

positive and the *SentimentText* columns containing the tweets in a raw form.



In the *Table 2.1.1* showing the first ten twitter posts we can already notice some

particularities and difficulties that we are going to encounter during the preprocessing steps.

● The presence of **acronyms** "bf" or more complicated "APL". Does it means apple ?

Apple (the company) ? In this context we have "friend" after so we could think that he

refers to his smartphone and so Apple, but what about if the word "friend" was not here ?

● The presence of **sequences of repeated characters** such as

"Juuuuuuuuuuuuuuuuussssst", "hmmmm". In general when we repeat several

characters in a word, it is to emphasize it, to increase its impact.

● The presence of **emoticons** , ":O", "T\_T", ":|"

and much more, give insights about user's

moods.

● **Spelling mistakes** and “ **urban grammar** ” like "im gunna" or "mi".

● The presence of **nouns** such as "TV", "New Moon".

Furthermore, we can also add,

● People also indicate their moods, emotions, states, between two *such as, \*\* cries\*,

\*hummin\*, \*sigh\*.

● The negation, “can't”, “cannot”, “don't”, “haven't” that we need to handle like: “I don’t like

chocolate”, “like” in this case is negative.

We could also be interested by the grammar structure of the tweets, or if a tweet is

subjective/objective and so on. As you can see, it is **extremely complex** to deal with languages

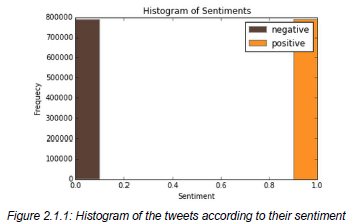
and even more when we want to analyse text typed by users on the Internet because people

don’t take care of making sentences that are grammatically correct and use a ton of acronyms

and words that are more or less english in our case.

We can visualize a bit more the dataset by making a chart of how many positive and

negative tweets does it contains,



We have exactly 790177 positive tweets and 788435 negative tweets which signify that the

dataset is wellbalanced.

There is also no duplicates.

Finally, let’s recall the Twitter terminology since we are going to have to deal with in the

tweets:

● Hashtag: A hashtag is any word or phrase immediately preceded by the # symbol. When

you click on a hashtag, you’ll see other Tweets containing the same keyword or topic.

● @username: A username is how you’re identified on Twitter, and is always preceded

immediately by the @ symbol. For instance, Katy Perry is @katyperry.

● MT: Similar to RT (Retweet), an abbreviation for “Modified Tweet.” Placed before the

Retweeted text when users manually retweet a message with modifications, for example

shortening a Tweet.

● Retweet: RT, A Tweet that you forward to your followers is known as a Retweet. Often

used to pass along news or other valuable discoveries on Twitter, Retweets always

retain original attribution.

● Emoticons: Composed using punctuation and letters, they are used to express emotions

concisely, ";) :) ...".

Now we have the corpus of tweets, we need to use other resources to make easier the

preprocessing

step.

**2.2 Resources**

In order to facilitate the preprocessing part of the data, we introduce five resources

which are,

● An **emoticon dictionary** regrouping 132 of the most used emoticons in western with

their sentiment, negative or positive.

● An **acronym dictionary** of 5465 acronyms with their translation.

● A **stop word dictionary** corresponding to words which are filtered out before or after

processing of natural language data because they are not useful in our case.

● A **positive and negative word dictionaries** given the polarity (sentiment outofcontext)

of words.

● A **negative contractions and auxiliaries dictionary** which will be used to detect

negation in a given tweet such as “don’t”, “can’t”, “cannot”, etc.

The introduction of these resources will allow to uniform tweets and remove some of their

complexities with the acronym dictionary for instance because a lot of acronyms are used in

tweets. The positive and negative word dictionaries could be useful to increase (or not) the

accuracy score of the classifier. The emoticon dictionary has been built from wikipedia with each

emoticon annotated manually. The stop word dictionary contains 635 words such as “the”, “of”,

“without”. Normally they should not be useful for classifying tweets according to their sentiment

but it is possible that they are.

Also we use Python 2.7 ( https://www.python.org/ ) which is a programming language

widely used in data science and scikitlearn

( http://scikitlearn.

org/ ) a very complete and useful

library for machine learning containing every techniques, methods we need and the website is

also full of tutorials wellexplained.

With Python, the libraries, Numpy ( http://www.numpy.org/ )

and Panda ( http://pandas.pydata.org/ ) for manipulating data easily and intuitively are just

essential.

**2.3 Pre-processing**

Now that we have the corpus of tweets and all the resources that could be useful, we

can preprocess the tweets. It is a very important since all the modifications that we are going to

during this process will directly impact the classifier’s performance. The preprocessing

includes cleaning, normalization, transformation, feature extraction and selection, etc. The result of

preprocessing will be consistent and uniform data that are workable to maximize the classifier's

performance.

All of the tweets are preprocessed by passing through the following steps in the same

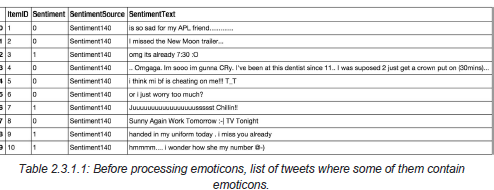
order.

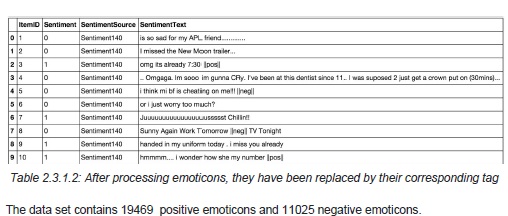
**2.3.1 Emoticons**

We replace all emoticons by their sentiment polarity **||pos||** and **||neg||** using the

emoticon dictionary. To do the replacement, we pass through each tweet and by using a regex

we find out if it contains emoticons, if yes they are replaced by their corresponding polarity.

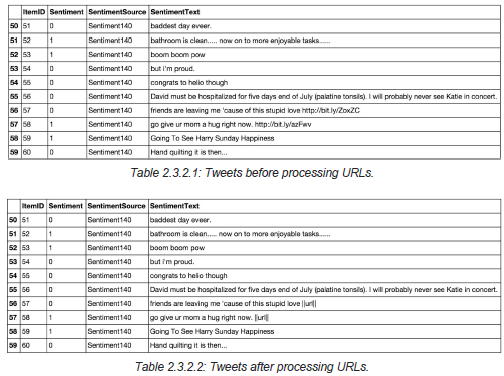




**2.3.2 URLs**

We replace all URLs with the tag **||url||** . There is about 73824 urls in the data set and we

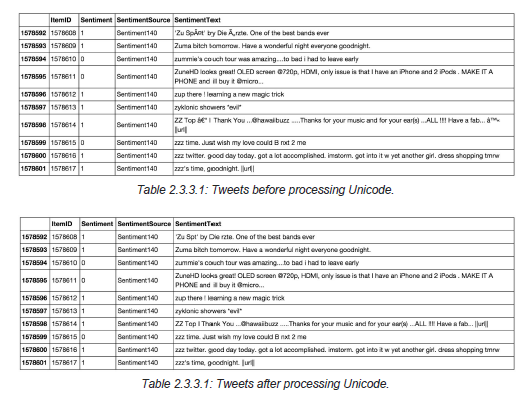
proceed as the same way we did for the emoticons.



**2.3.3 Unicode**

For simplicity and because the ASCII table should be sufficient, we choose to remove

any unicode character that could be misleading for the classifier.



**2.3.5 Case**

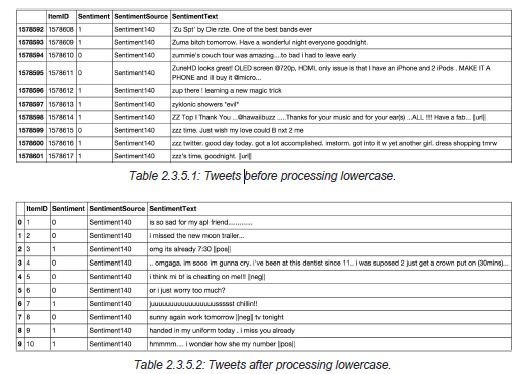
The case is something that can appears useless but in fact it is really important for

distinguish proper noun and other kind of words. Indeed: “General Motor” is the same thing that

“general motor”, or “MSc” and “msc”. So reduce all letters to lowercase should be normally done

wisely. In this project, for simplicity we will not take care of that since we assume that it should

not impact too much the classifier’s performance.

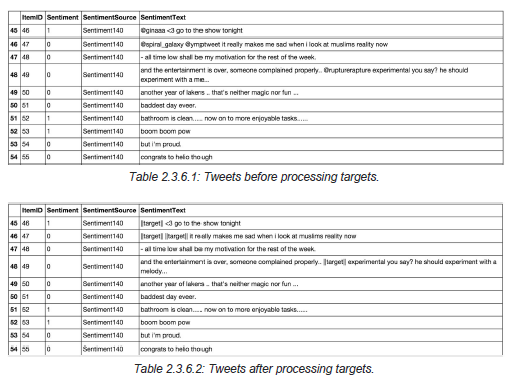


**2.3.6 Targets**

The target correspond to usernames in twitter preceded by “@” symbol. It is used to

address a tweet to someone or just grab the attention. We replace all usernames/targets by the

tag **||target||** . Notice that in the data set we have 735757 targets.



**2.3.7 Acronyms**

We replace all acronyms with their translation. An acronym is an abbreviation formed

from the initial components in a phrase or a word. Usually these components are individual

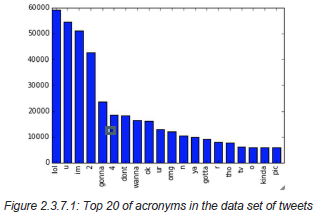
letters (as in NATO or laser) or parts of words or names (as in Benelux). Many acronyms are

used in our data set of tweets as you can see in the following bar chart.

At this point, tweets are going to be tokenized by getting rid of the punctuation and using

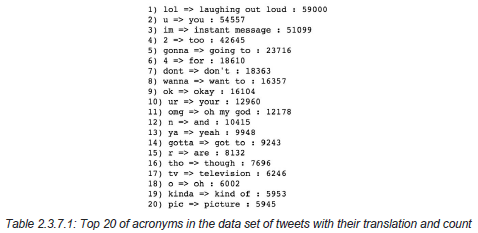
split in order to do the process really fast. We could use nltk.tokenizer but it is definitely much

much slower (also much more accurate).



As you can see, “lol”, “u”, “im”, “2” are really often used by users. The table below shows the top

20 acronyms with their translation and their count.



**2.4 Machine Learning**

Once we have applied the different steps of the preprocessing part, we can now focus

on the machine learning part. There are three major models used in sentiment analysis to

classify a sentence into positive or negative: SVM, Naive Bayes and Language Models

(NGram).

SVM is known to be the model giving the best results but in this project we focus only

on probabilistic model that are Naive Bayes and Language Models that have been widely used

in this field. Let’s first introduce the Naive Bayes model which is wellknown

for its simplicity and

efficiency for text classification.

**2.4.1 Naive Bayes**

In machine learning, naive Bayes classifiers are a family of simple probabilistic

classifiers based on applying Bayes' theorem with strong (naive)independence assumptions

between the features. Naive Bayes classifiers are highly scalable, requiring a number of

parameters linear in the number of variables (features/predictors) in a learning problem.

Maximumlikelihood

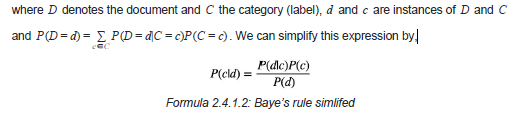
training can be done by evaluating a closedform

expression (mathematical

expression that can be evaluated in a finite number of operations), which takes linear time.

It is based on the application of the Baye’s rule given by the following formula:





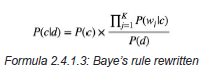
In our case, a tweet *d* is represented by a vector of *K* attributes such as *d* = (*w*1, *w*2, ..., *wk*) .

Computing *P*(*d*|*c*) is not trivial and that's why the Naive Bayes introduces the assumption that

all of the feature values *wj* are independent given the category label *c* . That is, for *i* =/ *j* , *wi* and

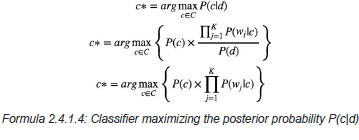
*wj* are conditionally independent given the category label *c* . So the Baye's rule can be rewritten

as,



Based on this equation, maximum a posterior (MAP) classifier can be constructing by seeking

the optimal category which maximizes the posterior *P*(*c*|*d*) :



Note that *P*(*d*) is removed since it is a constant for every category *c* .

There are several variants of Naive Bayes classifiers that are:

● The **Multivariate**

**Bernoulli Model** : Also called binomial model, useful if our feature

vectors are binary (e.g 0s and 1s). An application can be text classification with bag of

words model where the 0s 1s are "word does not occur in the document" and "word

occurs in the document" respectively.

● The **Multinomial Model** : Typically used for discrete counts. In text classification, we

extend the Bernoulli model further by counting the number of times a word $w\_i$

appears over the number of words rather than saying 0 or 1 if word occurs or not.

● the **Gaussian Model** : We assume that features follow a normal distribution. Instead of

discrete counts, we have continuous features.

For text classification, the most used considered as the best choice is the Multinomial Naive

Bayes.

The prior distribution *P*(*c*) can be used to incorporate additional assumptions about the

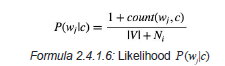
relative frequencies of classes. It is computed by:



where *N* is the total number of training tweets and *Ni* is the number of training tweets in class

*c* .

The likelihood is usually computed *P*(*wj*|*c*) using the formula:



where *count*(*wj*, *c*) is the number of times that word *wj* occurs within the training tweets of class

*c* , and |*V* | = Σ the size of the vocabulary. This estimation uses the simplest smoothing

*j*

*wj*

method to solve the **zeroprobability problem** , that arises when our model encounters a word

seen in the test set but not in the training set, **Laplace** or addone since we use 1 as constant.

We will see that Laplace smoothing method is not really effective compared to other smoothing

methods used in language models.

**2.4.2 Baseline**

In every machine learning task, it is always good to have what we called a baseline. It

often a “quick and dirty” implementation of a basic model for doing the first classification and

based on its accuracy, try to improve it.

We use the Multinomial Naive Bayes as learning algorithm with the Laplace smoothing

representing the classic way of doing text classification. Since we need to extract features from

our data set of tweets, we use the **bag of words model** to represent it.

The bag of words model is a simplifying representation of a document where it is

represented as a bag of its words without taking consideration of the grammar or word order. In

text classification, the count (number of time) of each word appears is a document is used as a

feature for training the classifier.

Firstly, we divide the data set into two parts, the training set and the test set. To do this,

we first shuffle the data set to get rid of any order applied to the data, then we from the set of

positive tweets and the set of negative tweets, we take 3/4 of tweets from each set and merge

them together to make the training set. The rest is used to make the test set. Finally the size of

the training set is 1183958 tweets and the test set is 394654 tweets. Notice that they are

balanced and follow the same distribution of the initial data set.

Once the training set and the test set are created we actually need a third set of data

called the **validation set** . It is really useful because it will be used to **validate our model**

**against unseen data and tune the possible parameters** of the learning algorithm to avoid

underfitting and overfitting for example. We need this validation set because our test set should

be used only to verify how well the model will **generalize** . If we use the test set rather than the

validation set, our model could be **overly optimistic and twist the results** .

To make the validation set, there are two main options:

● Split the training set into two parts (60%, 20%) with a ratio 2:8 where each part contains

an equal distribution of example types. We train the classifier with the largest part, and

make prediction with the smaller one to validate the model. This technique works well

but has the disadvantage of our classifier not getting trained and validated on all

examples in the data set (without counting the test set).

● The **Kfold crossvalidation** . We split the data set into k parts, hold out one, combine

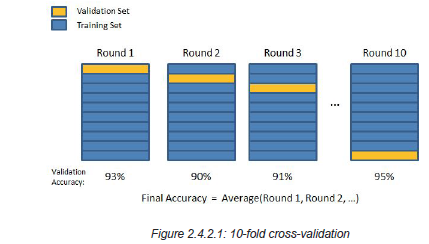
the others and train on them, then validate against the heldout

portion. We repeat that

process k times (each fold), holding out a different portion each time. Then we average

the score measured for each fold to get a more accurate estimation of our model's

performance.



We split the training data into 10 folds and cross validate on them using scikitlearn as shown in the figure 2.4.2.1 above. The number of Kfolds is arbitrary and usually set to 10 it is

not a rule. In fact, determine the best K is still an unsolved problem but with lower K:

computationally cheaper, less variance, more bias. With large K: computationally expensive,

higher variance, lower bias.

We can now train the naive bayes classifier with the training set, validate it using the

hold out part of data taken from the training set, the validation set, repeat this 10 times and

average the results to get the final accuracy which is about **0.77** as shown in the screen results

below,



*Figure 2.4.2.2: Result of the naive bayes classifier with the score representing the average of*

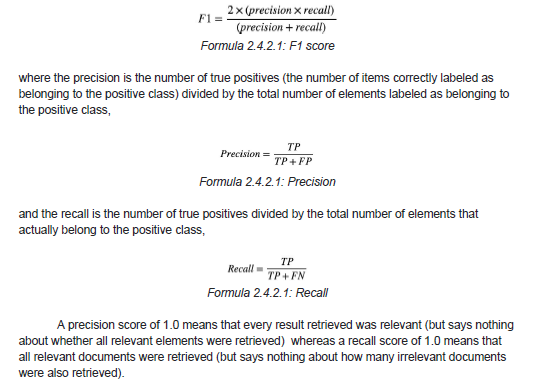
*the results of each 10fold crossvalidation, and the overall confusion matrix.*

Notice that to evaluate our classifier we two methods, the F1 score and a confusion

matrix. The **F1 Score** can be interpreted as a weighted average of the precision and recall,

where an F1 score reaches its best value at 1 and worst score at 0. It a measure of a

**classifier's accuracy** . The F1 score is given by the following formula,



There is a **tradeoff between precision and recall** where increasing one decrease the

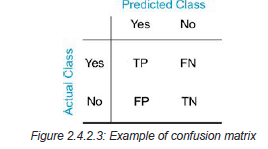
other and we usually use measures that combine precision and recall such as Fmeasure or MCC.

A **confusion matrix** helps to visualize how the model did during the classification and

evaluate its accuracy. In our case we get about 156715 false positive tweets and 139132 false

negative tweets. It is "about" because these numbers can vary depending on how we shuffle our

data for example.



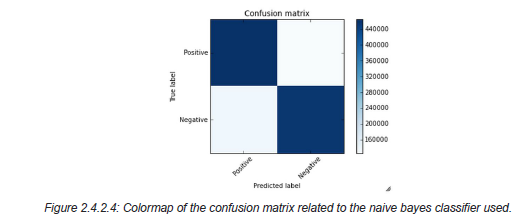
Notice that we still didn't use our test set, since we are going to tune our classifier for

improving its results.

The confusion matrix of the naive bayes classifier can be expressed using a color map

where dark colors represent high values and light colors represent lower values as shown in the

corresponding color map of the naive bayes classifier below,



Hopefully we can distinguish that the number of true positive and true negative classified tweets

is higher than the number of false and positive and negative tweets. However from this result we

try to improve the accuracy of the classifier by experimenting different techniques and we repeat

the same process using the kfold

cross validation to evaluate its averaged accuracy.

**2.4.3 Improvements**

From the baseline, the goal is to improve the accuracy of the classifier, which is 0.77, in

order to determine better which tweet is positive or negative. There are several ways of doing

this and we present only few possible improvements (or not).

First we could try to removed what we called, stop words. Stop words usually refer to the

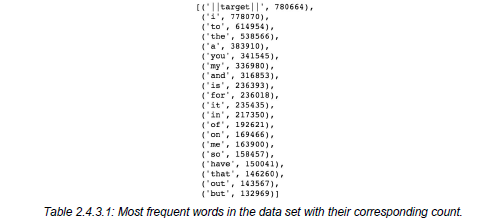
most common words in the English language (in our case) such as: "the", "of", “to” and so on.

They do not indicate any valuable information about the sentiment of a sentence and it can be

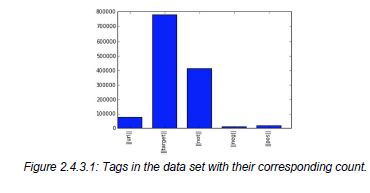
necessary to remove them from the tweets in order to keep only words for which we are

interested. To do this we use the list of 635 stopwords that we found. In the table below, you

can see the most frequent words in the data set with their counts,



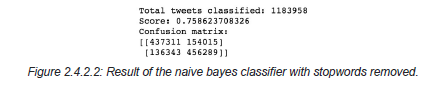
We can derive from the table, some interesting statistics like the number of times the tags used in the preprocessing step appear,



Recall that ||url|| corresponds to the URLs, ||target|| the twitter usernames with the symbol “@”

before, ||not|| replaces the negation words, ||pos|| and ||neg|| replace the positive and negative

smiley respectively. After removing the stop words we get the results below,



Compared to the previous result, we lose in accuracy and the number of 0.02 false positive goes

from 126305 to 154015. We conclude that stop words seem to be useful for our classification

task and remove them do not represent an improvement.

We could also try to stem the words in the data set. **Stemming** is the process by which

endings are removed from words in order to remove things like tense or plurality. The stem form

of a word could not exist in a dictionary (different from Lemmatization). This technique allows to

unify words and reduce the dimensionality of the dataset. It's not appropriate for all cases but

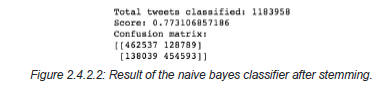
can make it easier to connect together tenses to see if you're covering the same subject matter.

It is faster than **Lemmatization** ( remove inflectional endings only and return the base or

dictionary form of a word, which is known as the lemma ). Using the library NLTK which is a

library in Python specialized in natural language processing, we get the following results after

stemming the words in the data set,



We actually lose 0.002 in accuracy score compared to the results of the baseline. We conclude

that stemming words does not improve the classifier’s accuracy and actually do not make any

sensible changes.

**2.4.4 Language Models**

Let’s introduce language models to see if we can have better results than those for our

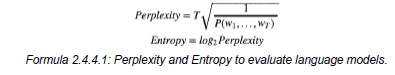
baseline. Language models are models assigning **probabilities to sequence of words** .

Initially, they are extensively used in speech recognition and spelling correction but it turns out

that they give good results in text classification.

The quality of a language model can be measured by the empirical perplexity (or

entropy) using:



The goal is to **minimize the perplexity which is the same as maximizing probability** .

An **Ngram model** is a type of probabilistic language model for predicting the next item

in such a sequence in the form of (n 1)

order Markov Model. The Markov assumption is the

probability of a word depends only on the probability of a limited history (previous words).



A straightforward maximum likelihood estimate of ngram probabilities from a corpus is given by

the observed frequency,



There are several kind of ngrams but the most common are the unigram, bigram and trigram.

The **unigram model** make the assumption that every word is independent and so we compute

the probability of a sequence using the following formula,



In the case of the **bigram model** we make the assumption that **a word is dependent of its**

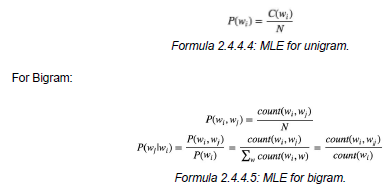
**previous word** ,



To estimate the ngram probabilities, we need to compute the **Maximum Likelihood**

**Estimates** .

For Unigram:



Where N is the number of words, means count, *N C wi* and *wj* are words.

There are two main practical issues:

● We compute everything in log space (log probabilities) to avoid underflow (multiplying so

many probabilities can lead to too small number) and because adding is faster than

multiplying (*p*1 × *p*2 × *p*3) = (*logp*1 *log* ) + *p*2 + *logp*3

● We use smoothing techniques such as Laplace, WittenBell

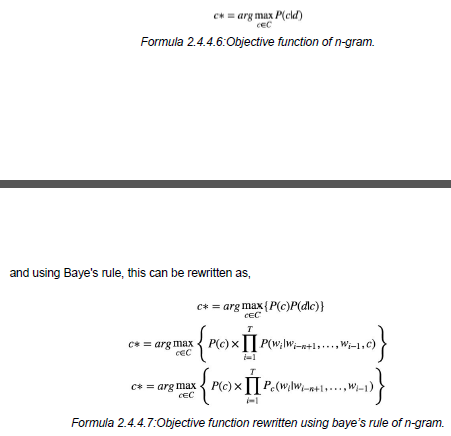
Discounting, GoodTuring

Discounting to deal with unseen words in the training occurring in the test set.

An Ngram

language model can be applied to text classification like Naive Bayes model does. A

tweet is categorized according to,



*P*(*d*|*c*) *d c* an ngram language model.

**An important note is that ngram classifiers are in fact a generalization of Naive Bayes. A**

**unigram classifier with Laplace smoothing corresponds exactly to the traditional naive**

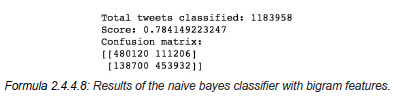
**Bayes classifier.**

Since we use bag of words model, meaning we translate this sentence: "I don't like chocolate"

into "I", "don't", "like", "chocolate", we could try to use bigram model to take care of negation

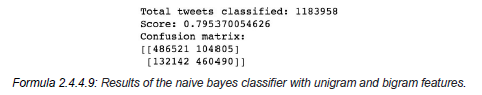
with "don't like" for this example. Using bigrams as feature in the classifier we get the following

results,



Using only bigram features we have slightly improved our accuracy score about 0.01. Based on

that we can think of adding unigram and bigram could increase the accuracy score more.



and indeed, we increased slightly the accuracy score about 0.02 compared to the baseline.

3. Conclusion

Nowadays, sentiment analysis or opinion mining is a hot topic in machine learning. We

are still far to detect the sentiments of s corpus of texts very accurately because of the

complexity in the English language and even more if we consider other languages such as

Chinese.

In this project we tried to show the basic way of classifying tweets into positive or

negative category using Naive Bayes as baseline and how language models are related to the

Naive Bayes and can produce better results. We could further improve our classifier by trying to

extract more features from the tweets, trying different kinds of features, tuning the parameters of

the naïve Bayes classifier, or trying another classifier all together.