University of Passau

Text Mining Project

Classification of Micro Blogging Text

Author:
Haythem Sahbani

Supervisor:
Dr. Behrang Q. Zadeh

February 2015

0. Contents

0. List of Figures

0. List of Tables

1. Introduction

Text mining is used more and more to improve the way we get information or to enhance the way we perceive it. It is incorporated in many fields like politics, social media or improve existing technologies like spam filtering. Text classification is a branch of text mining. It is a classic task in natural language processing. Given a text document and a set of categories, the text classification algorithm must assign the text document into one or more of the given text categories.
Nowadays social networks became a pillar for social interaction and Micro blogging is a new form of communication in which users can describe their current status in short posts distributed by instant messages, mobile phones, email or the Web. Twitter, a popular micro blogging tool has seen a lot of growth since it launched in October, 2006. It helps users to connect with their followers. The tweets from users are referred to as micro blogs because there is a 140 character limit imposed by Twitter for every tweet. This lets the users present any information with only a few words, optionally followed with a link to a more detailed source of information. The project title in the domain of text classification, more precisely the task of short text classification, which is more challenging than classic document classification. In the proposed project title, we use the Twitter Political Corpus"[1]. Given a tweet, the task is to distinguish political tweets from non-political ones.

1.1 Motivations and goals

A variety of techniques for supervised learning algorithms have demonstrated reasonable performance for text classification. The question is how would they perform for short text classification. The project aims to fetch different classification settings, techniques and show how they perform in micro blog text classifications. Three different classifiers will be tested: Naïve Bayes, Maximum Entropy and Support Vector Machine.

1.2 Outline

Figure 1.1 shows the process undertaken in this project. First we will see how the raw corpus is processed. Then how are the features extraction. Finally the classification results and discussions.

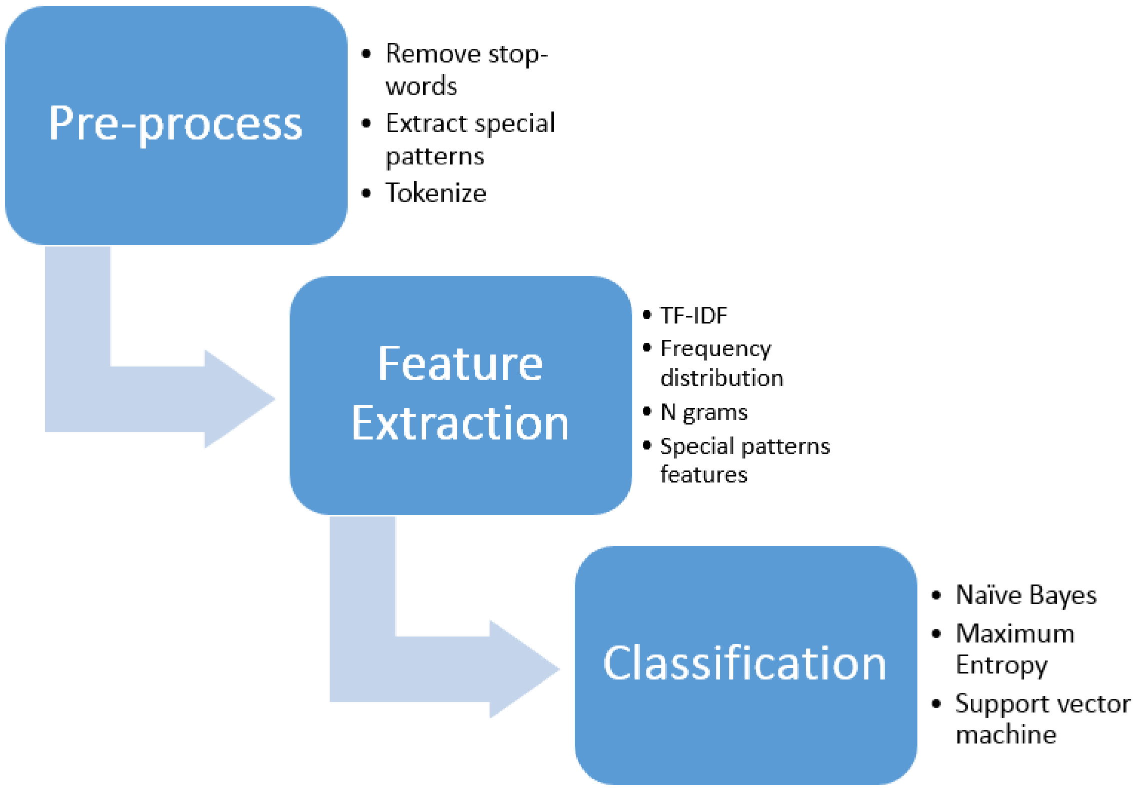

FIGURE 1.1: Process chain

2. Data set Processing

Raw data is generally not well fitted for the needs of a project. This can be driven by the lack of attributes values or certain attributes of interest. Also real world data is noisy. It contains errors or outliers. Therefore the data needs to be processed. The next sections will show how the raw text is processed to get the most meaningful features out from it. These features will be next used in the classification task. But first, a brief presentation of the data set is mandatory.

2.1 The data set

The data set[1] provides resources to develop learning algorithms that link political statements on Twitter to general opinions about government and politicians. It is composed of two files of tweets that have been hand labeled for their topics, specifically, discussing politics or not discussing politics. Each file contains about 2000 tweets, one tweet per

line. A line contains two fields separated by a single tab character: the label, and the text of the tweet:
POLIT RT @AdamSmithInst Quote of the week: My political opinions lean more and more towards Anarchy
NOT @DeeptiLamba LOL, I like quotes. Feminist, anti-men quotes.
The first file is a randomly selected set of 2000 tweets from Twitter's "spritzer" feed collected between June 1, 2009 and Dec 31, 2009. The second corpus is not selected from the entire feed, but rather randomly selected from a subset of tweets that contained at least one political keyword in each tweet. The two labels are *POLIT* (political) and *NOT* (not political).
The two files are merged in the context of this work. In fact, on one side the *Politics General Tweet Corpus* contains about 90% of tweets labeled as *non-political.* On the other side the *Politics Keyword Tweet Corpus* contains about 90% of tweets labeled as *political.* If let so the classifier will have a good accuracy but in reality it won't do any classification task since 90% of its entries are about one label. By combining both corpora, the labels are equally distributed and a classifier can learn from it to predict political and non-political tweets. The two corpora are combined in the **_tweets.txt** file.

2.2 Processing

This section explains the assumptions and techniques used to process raw tweets.
For further testing and manipulations, the python code is available in the **preprocess.py** file.

2.2.1 Special tweet patterns

Since in tweets there are many links, hashtags[1], users[2] and 'retweet' tokens, they need to be processed in a specific way. At first, I made a hypothesis that links and 'retweet' tokens are relevant for political classification. This assumption was made on these basis:

- Political tweets are important and thus they are highly 'retweeted'
- Political tweets refer to important matters and tweets are limited in 140 characters. That's why they include links to explain the subject in more details.

Unfortunately after seeing the distribution of these patterns in political and non-political tweets, it appears that they are equally scattered. As a result links and "retweet" tokens don't add any information Therefore they have been deleted in the process.

In second place, hashtags and users are unique so meaningful for classification purpose.

[1]Words that begin with #
[2]Words that begin with @

2.2.2 Delete stop words

Not all words in a language are representative of a topic or a sentiment analysis. As a consequence they don't add information in the classification task. These words are called stop words[2]: 'They are words which are filtered out before or after processing of natural language data (text). There is no single universal list of stop words used by all processing of natural language tools, and indeed not all tools even use such a list'. Therefore a stop word list[3] is created to fulfill the purpose of this work.
Generally English words are contracted. For example: *'do not'* is written *'don't'*. To prevent adding more entries to the stop word list, a contraction list[4] that maps contracted words to their standard form has been created to expand these contractions.

2.2.3 Tokenize tweets

The tokenization of tweets is processed using the *nltk regexp tokenizer*[5]. This tool tokenizes text using a regular expression pattern. The most important patterns that are implemented in the tokenizer are `(\@\w*)+` and `(\#\w*)+` as they prevent the tokenizer from deleting the # and @ symbols since they are important for the classification part.

After processing the data, tweets that have 1 or less token are deleted.

2.3 Feature extraction

The final step for processing the data is feature extraction. This part is influenced by the machine learning algorithms that will be used to classify the tweets in the next step. In this way, feature engineering is really important and it strongly affects the final results. Also, this part is revisited as the project work progressed.
A bag of words feature set is used in this project. Further more part of speech tagging is not used because tweet text is not formal and too sparse.
For further testing and manipulations, the python code is available in the **feature_extraction.py** file.

2.3.1 Term Frequency - Inverse Document Frequency (tf-idf) features

tf-idf [3] is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in

[3]stop_words.py file
[4]contractions.py file
[5]URL: RegexpTokenizer

general.
A *scikit learn TfidfVectorizer*[6] is used to extract the *tf-idf* feature set.

2.3.2 Unigram frequency distribution features

A bag of words feature set using frequency distribution counter is used from the *nltk FreqDist*[7] class.

2.3.3 Tweets special patterns features

As mentioned in section (2.2.1) hashtags and users are unique and therefore they are relevant features for the classification task. However, not all the hashtags and users occur often in the data set. Moreover, not all tweets contain at least one hashtag or one user name. In fact the data set contains 3259 tweets without hashtags and 2224 without user names. These designate respective rates of 84% and 57% of the tweets. In this way only the most common hashtags and users are used as features. Also, these features are combined with a bag of words feature set to have a better impact on the classification process.
The hashtag and users extractor can be tested on the **feature_extraction.py** file.

2.3.4 Bigram frequency distribution features

A bigram is a pair of consecutive written units such as letters, syllables, or words. The bigrams are extracted using the *nltk bigram*[8] module.

3. Classification Algorithms

Machine learning algorithms are the fundamental of all classification program. These algorithms are divided in two big families: the supervised learning and the unsupervised learning. In this project only supervised learning models are used. This section presents briefly the classification algorithms used in this work.

[6]URL: TfidfVectorizer
[7]URL: FreqDist
[8]URL: Collocations

3.1 Naïve Bayes classifier

A Naïve Bayes classifier[4] is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. 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. Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting.

3.1.1 The Naïve Bayes probabilistic model

Abstractly, the probability model for a classifier is a conditional model.

$$P(C|F_1, \cdots, F_n)$$

Over a dependent class variable C with a small number of outcomes or *classes*, conditional on several feature variables F_1 through F_n . The problem is that if the number of features n is large or when a feature can take on a large number of values, then basing such a model on probability tables is infeasible. Therefore the model is formulated to make it more tractable using Bayes' theorem:

$$P(C|F_1, \cdots, F_n) = \frac{p(C)p(F_1, \cdots, F_n|C)}{p(F_1, \cdots, F_n)}$$

The equation could be formulated as follow in a literal way

$$posterior = \frac{prior * likelihood}{evidence}$$

3.2 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. The figure 3.1 shows an example for a linearly separable set of 2D-points which belong to one of two classes.

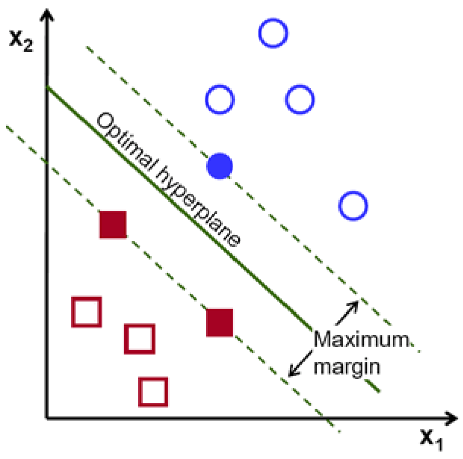

FIGURE 3.1: Optimal hyperplane

The optimal hyperplane computedusing equation 3.1:

$$f(x) = \beta_0 + \beta^T x \tag{3.1}$$

Where β is known as the weight vector and β_0 as the bias.

3.3 Maximum Entropy

The maximum entropy classifier estimates probabilities based on the principle of making as few assumptions as possible, other than the constraints imposed. Such constraints are derived from training data, expressing some relationship between features and outcome. The probability distribution that satisfies the above property is the one with the highest entropy. It is unique, agrees with the maximum-likelihood distribution, and has the exponential form :

$$p(o|h) = \frac{1}{Z(h)} \prod_{j=1}^{k} \alpha_j^{f_j(h,o)} \tag{3.2}$$

Where:

- o refers to the outcome
- h refers to the history or context.
- $Z(h)$ is a normalization function.
- $f_j(h,o)$ is a binary function.
- α_j is estimated by a procedure called Generalized Iterative Scaling (GIS). This is an iterative method that improves the estimation of the parameters at each iteration.

4. Results and Discussion

The evaluation task is the most crucial step for a project. This section explains how this process is conducted then discuss the results. But first, let's take a look at the standard machine learning evaluations methods.

4.1 Machine learning evaluation methods

In pattern recognition and information retrieval with binary classification usually a confusion matrix is used to evaluate the systems performance. A confusion matrix *(Kohavi and Provost, 1998)* contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix.

Theorem 4.1. Confusion Matrix

$$\begin{array}{cc} & \begin{array}{cc} political & non-political \end{array} \\ \begin{array}{c} political \\ non-political \end{array} & \begin{pmatrix} Tp & Fn \\ Fp & Tn \end{pmatrix} \end{array}$$

Where:

- Tp (True positive): is the number of **correct** predictions that an instance is political.
- Fn (False negative): is the number of **incorrect** predictions that an instance is non-political.
- Fp (False positive): is the number of **incorrect** of predictions that an instance political.
- Tn (True negative): is the number of **correct** predictions that an instance is non-political.

From the Confusion Matrix 4.1, different evaluation patterns are extracted:

$$Accuracy = \frac{Tp + Tn}{Tp + Tn + Fp + Fn} \tag{4.1}$$

$$Precision = \frac{Tp}{Tp + Fp} \tag{4.2}$$

$$Recall = \frac{Tp}{Tp + Fn} \tag{4.3}$$

$$F_1 = 2.\frac{precision.recall}{precision + recall} = \frac{2Tp}{2Tp + Fp + Fn} \tag{4.4}$$

To balance the influence between precision and recall in traditional text classification, we will adopt the F_1-Measure4.4 as our precision's criterion to evaluate the performance of the classifiers in a global aspect.

4.2 Evaluation and Results

First, the corpus is divided into a *test set* and a *training set* where the size of each one is respectively $\frac{3}{4}.corpus$ and $\frac{1}{4}.corpus$.

Then to model a real life situation the features are extracted only from the training set.

The evaluation aim is to answer the following questions: combine the following criterion to get the best classification settings:

- What is the optimal number of features for a classifier?
  As the Maximum Entropy classifier needs a lot of time to learn, the maximum number of feature tested is capped at 200.
- How does the different machine learning algorithms perform?
- What is the best feature engineering pattern ?

The following sections will answer the above questions. In addition for further testing and manipulations, the python code is available in the **evaluation.py** file.

4.2.1 Unigram feature

Special patterns features2.3.3 (hashtags and names) are extracted using the *Patterns-Features()* class in the **feature_extraction.py** file. The 10 most frequent names and hashtags are taken. They are added to the bag of word feature set. The results of the next subsections can be reproduced by running the *unigram_evaluation(lines)* function in the **evaluation.py** file.

4.2.1.1 Frequency distribution

Figure 4.1 shows the results of the evaluation using the Frequency distribution2.3.2 feature selection. In fact, we can see that the F_1-Measure of the algorithms are relatively close for a certain number of features. Table 4.1 summarizes the results.

	max f-measure	number of features
Naive Bayes	0.891013	190
Maximum entropy	0.893058	70
Support Vector Machine	0.900000	110

Table 4.1: Best algorithms results using Frequency Distribution

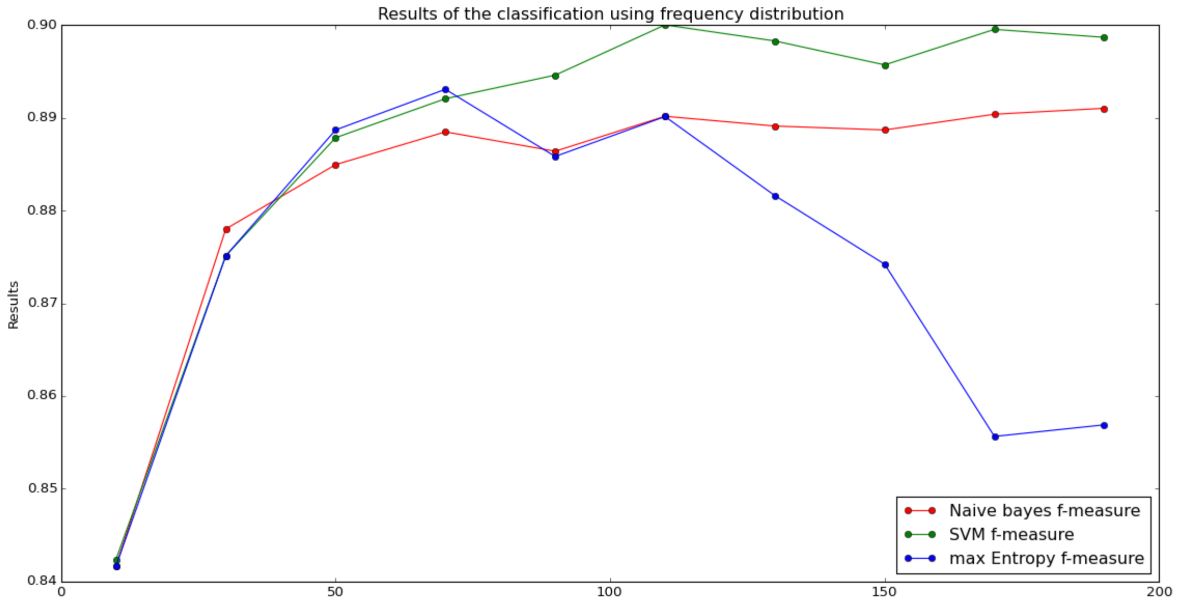

Figure 4.1: Frequency Distribution Unigram F-Measure results for different number of features

4.2.1.2 TF-IDF evaluation

Figure 4.2 shows the results of the evaluation using the TF-IDF2.3.1 feature selection. In fact, we can see that the F_1-Measure of the algorithms are relatively close for a certain number of features. Table 4.2 summarizes the results.

	max f-measure	number of features
Naive Bayes	0.894837	190
Maximum entropy	0.892791	90
Support Vector Machine	0.904215	190

Table 4.2: Best algorithms results using TF-IDF

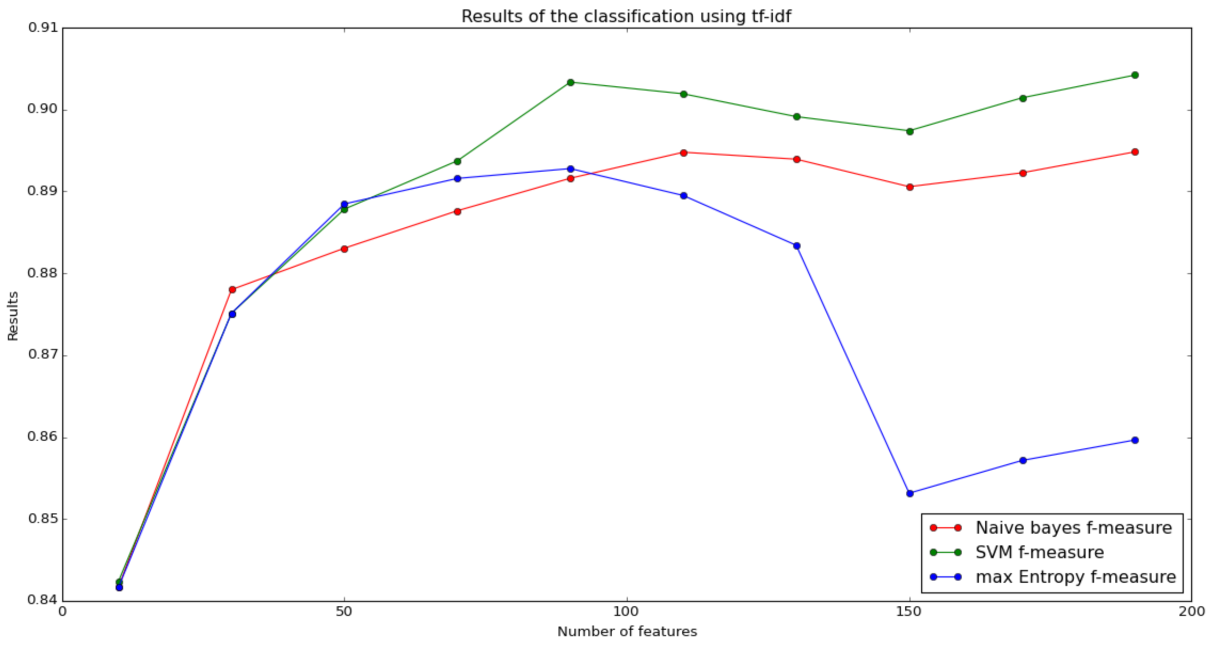

FIGURE 4.2: tf-idf unigram F-Measure results for different number of features

4.2.2 Bigram features

Figure 4.3 shows that results of Bigram2.3.4 features are constant. To have a better explanation of the result, the accuracy has been taken as well as the F_1-Measure. Table 4.2 summarizes the results.

	max f-measure	number of features
Naive Bayes	0.754170	170
Maximum entropy	0.752347	170
Support Vector Machine	0.753623	170

TABLE 4.3: Best algorithms results using Frequency Distribution

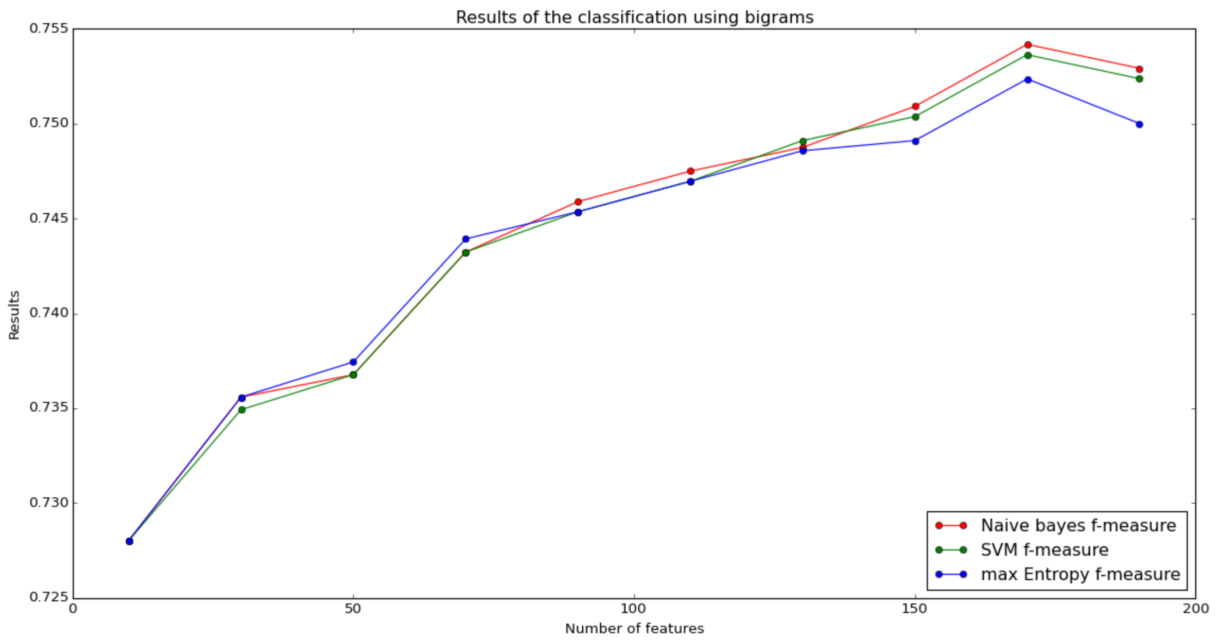

Figure 4.3: Bigrams F-Measure results for different number of features

The results of this section can be reproduced by running the *bigram_evaluation(lines)* function in the **evaluation.py** file.

4.2.3 Combination of Unigrams and Bigrams

The results shown in section 4.2.2 inform us that the overall outcome is independent from the number of bigrams used. At this end, only the 20 most frequent bigrams are used for this section.
Figure 4.4 shows the results of the evaluation using Bigrams and Frequency Distribution feature selection. Table 4.2 summarizes the results.

	max f-measure	number of features
Naive Bayes	0.818182	190
Maximum entropy	0.812893	150
Suport Vector Machine	0.824576	200

Table 4.4: Best algorithms results using Unigrams and Bigrams

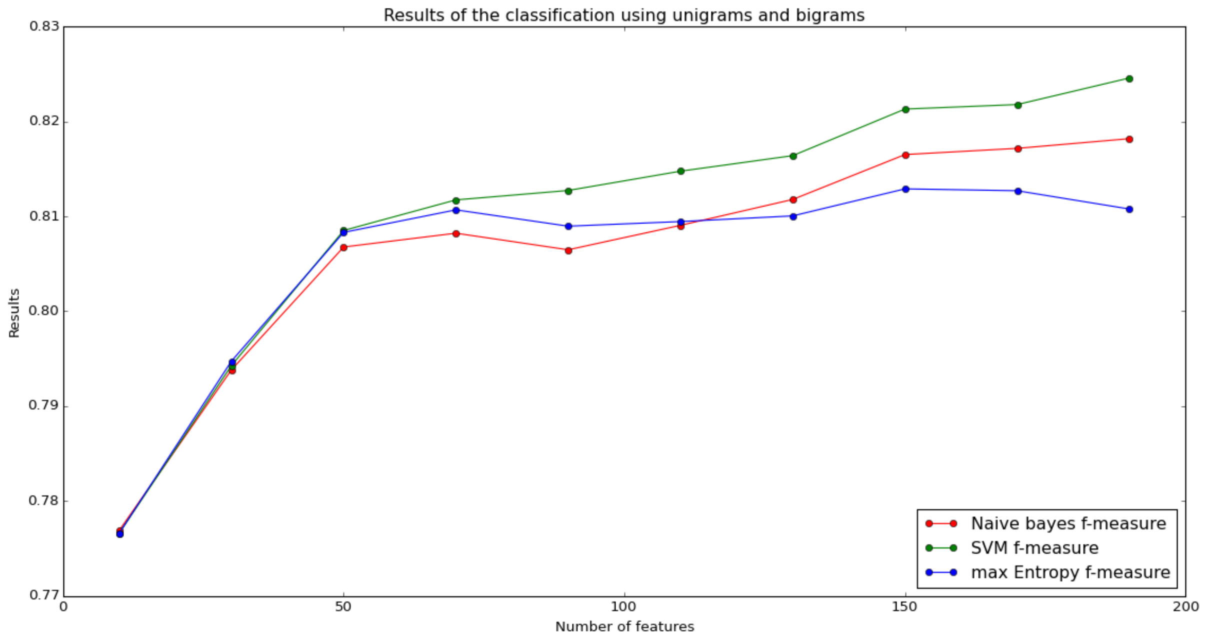

Figure 4.4: Combination of Unigrams and Bigrams F-Measure results for different number of features

The results of this section can be reproduced by running the *uni_and_bi_evaluation(lines)* function in the **evaluation.py** file.

4.3 Discussions

4.3.1 Result discussion

The F_1-Measure results are relatively close in each test case. We can see that the Support Vector Machine classifier has a slight edge.
The unigram classification test the results show that the Support Vector Machine and Naïve Bayes classifiers reaches their best F_1-Measure for 190 features. This is an expected result because of the nature of these classifiers. However the maximum entropy falls down in F_1-Measure after reaching a peak at 70 features. This is explained by the motivation behind maximum entropy. In fact it says that one should prefer the most uniform models that satisfy any given constraint. By adding more features, the maximum entropy classifier needs more iterations to reach the best results. In this work the number of iterations for the Maximum Entropy classifier is 100 as it takes a lot of computation time.
The bigram classification test the results display an inferior result comparing to the unigram findings. This is explained by the frequency of a bigram. In fact the frequency is very low compared to the unigrams.
This can be verified by running the next code.

```
import preprocess
```

```
from nltk import FreqDist, bigrams
filename = "all_tweets.txt"
lines = preprocess.main(filename)
all_tweets = " ".join([" ".join(line[1]) for line in lines])
print " 10 most frequent bigrams", FreqDist(bigrams(all_tweets.split(" ")))
    .most_common(10)
print " 10 most frequent unigrams", FreqDist(all_tweets.split(" ")).
    most_common(10)
```

The outcome of the code of shown above reveal that the most frequent bigram appears 55 times although the most frequent unigram appears 429 times.

```
10 most frequent bigrams [((u'health', u'care'), 55), ((u'sarah', u'palin'
    ), 37), ((u'president', u'obama'), 28), ((u'barack', u'obama'), 22), ((
    u'white', u'house'), 16), ((u'care', u'reform'), 16), ((u'bill', u'
    clinton'), 16), ((u'#tcot', u'#tlot'), 16), ((u'pres', u'obama'), 13),
    ((u'blog', u'post'), 13)]
 10 most frequent unigrams [(u'obama', 433), (u'government', 260), (u'
    economy', 177), (u'afghanistan', 126), (u'2', 120), (u'good', 117), (u'
    senate', 115), (u'#tcot', 114), (u'congress', 110), (u'lol', 104)]
```

Finally, the combination of unigrams and bigrams has a fair result but not as good as unigram alone.

4.3.2 Limitations

The data set was collected between June 1, 2009 and Dec 31, 2009. The political subjects are dynamic and topics, key words change often. Therefor this model can't classify real time tweets.
Moreover this classification model works only on English tweets.
The special patterns features are limited since more than 50% of the tweets does not contain a hashtag or/ and name.

4.3.3 Improvements

This project work can be improved, these are some ways:

- Test unsupervised learning algorithms.
- Try to fetch new political tweets to keep the corpora up to date.

5. Conclusion

In this study we have analyzed a large social network in a new form of social media known as micro blogging. The work showed results for some classic classification techniques applied to short text classification. These results reveal that Support Vector Machine has the best F_1-Measure but the Maximum entropy needs less features than the Support Vector Machine to get almost the same outcome. Moreover, the unigrams features are better for this kind of classification.

5. Bibliography

[1] The twitter political corpus, February 2015. URL `http://www.usna.edu/Users/cs/nchamber/data/twitter/`.

[2] Wikipedia. Stop words, January 2015. URL `http://en.wikipedia.org/wiki/Stop_words`.

[3] Wikipedia. tf–idf, February 2015. URL `http://en.wikipedia.org/wiki/Tf%E2%80%93idf`.

[4] Naive bayes classifier, February 2015. URL `http://www.ic.unicamp.br/~rocha/teaching/2011s2/mc906/aulas/naive-bayes-classifier.pdf`.

[5] Steven Bird, Erwan Klein, and Edward Loper. *Natural Language Processing with Python*. O'REILLY.

[6] Kamal Nigam, John Laffety, and Andrew McCallum. Using maximum entropy for text classification. Technical report, School of computer Science Carnegie Mellon University Pittsburgh.

[7] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: A maximum entropy approach using global information. Technical report, DSO National Laboratories. Department of Computer Science School of Computing National University of Singapore.

[8] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: Understanding microblogging usage and communities. Technical report, University of Maryland Baltimore County, NEC Laboratories America.

[9] Zitao Liu, Wenchao Yu, Wei Chen, Shuran Wang, and Fengyi Wu. Short text feature selection for micro-blog mining. Technical report, International School of Software Wuhan University Wuhan, China. School of Electronics and Information Engineering Sichuan University Chengdu, China.

[10] Julie Kane Ahkter and Steven Soria. Sentiment analysis: Facebook status messages. Technical report.

[11] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. Technical report, Stanford University.