Sentiment Probing of Social Media Data using Various Supervised Learners

Sourangshu Das and S. Kami Makki
Department of Computer Science, Lamar University, Beaumont, Texas, USA

Abstract - the rise in usage of Facebook, Myspace, Twitter, blogs, forums and other online means of communication has created a digital landscape where people are able to post their opinions or socialize through a variety of means and applications. Analyzing these online postings using sentiment analysis algorithms facilitates a general overview about different matters. As sentiment analysis is the computational study of opinions, evaluations, attitudes and emotions expressed by people in their posts. We have applied Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Logistic Regression and Linear Support Vector classifiers on training dataset gathered from online postings. The datasets are already labelled for negative or positive texts and with the help of Bag-of-words and TF-IDF methods we have conducted the feature extraction from the texts of each type and trained the above classifiers on those particular features. Our study shows that the classifiers perform differently depending on the training data, and the increase in the dataset size will improve their accuracy, as well as ascertains a general trend about people’s opinion on a particular subject.

Keywords: Sentiment Analysis; Feature Selection; Classification; Learning Algorithms; Social Media data

1 Introduction

Sentiment analysis is the computational study of opinions, evaluations, sentiments, attitudes and emotions expressed by people in their posts. These reviews could be about a specific product or an independent event. With increasing blogging websites, a huge number of reviews are available daily online. Sentiment Analysis makes the user aware of the positive and negative aspect of a product or event. This methodology can also be used by companies to take feedback from the users about different products. For Example, the recent natural disaster Hurricane Harvey has caused billions of dollars of damage, which is more than any natural disaster in U.S history. It affected millions of people from Texas through Louisiana, Mississippi, Tennessee, and Kentucky. People all over these states have different opinion about the damages caused to their localities and how quickly different private and governmental organizations responded to this disaster. Through analyzing the tweets from people across the country we can extract an overview on how they are reacting to the steps taken by different governmental organizations. Or after any other tragic events across the country such as the recent event of Las Vegas shootings a number of ideas presented as to how best handle similar future tragic events.

Sentiment analysis considers the opinions only and is based on the idea that an opinion consists of an emotion which could be either positive or negative. Extraction of features from text, determining the polarity of opinion and grouping the feature synonyms, are the main steps in sentiment analysis process. As Feature extraction does not consider the proper language constructs (paragraphs, sentences, grammars, clauses or phrases), we have taken the classification algorithms such as Bernoulli Naïve Bayes, Multinomial Naïve Bayes, Linear Support Vector Classification, Logistic Regression into consideration to identify which of the algorithm provides better classification for analyzing people opinion. This study is focused on a comparative analysis of these algorithms.

2 Literature Review

2.1 Background

There are numerous research papers and studies that focus on sentiment classification for Twitter. The studies have described many interesting methodologies of detecting and identifying sentiment from Twitter data. But it has been seen that the informal tone of tweets poses a challenge for analysis. Sentiment analysis has provided a way to a wide range of research areas ranging from document level classification to sentence level [9] leading to phrases and finally feature extraction.

Classifying tweets into positive and negative using distant supervision was presented by Alec Go, Richa Bhayani and Lei Huang [2]. They presented an approach for classifying tweets with respect to a query term. Use of linguistic features for detecting sentiment of twitter messages was investigated by Efthymios Kouloumis, Theresa Wilson and Johanna Moore [7]. They used hash-tagged dataset (HASH) for development and training. Yeqing Yan, Hui Yang and Hui-ming Wang [16] have worked on classifiers like Multinomial Naïve Bayes, MaxEnt and SentiStrength to build an effective ensemble classifier which can reach to be the state of the art in case of tweet classification. They put the classifiers under careful study to determine whether they are complimentary with each other using hand-annotated Twitter datasets. No additional feature engineering was done, and the result showed their
ensemble classifier to be the top performers among all of the above classifiers. Harsha Sinha and Arashdeep Kaur [8] in their survey and comparative study of sentiment analysis algorithms have given a comparison between a number of sentiment analysis algorithms according to their accuracy. This paper helps in identifying the algorithm suited for a particular dataset.

2.2 Text Feature Extraction

We have used Python 3.6 as the programming language for its ease of use due to simple syntax. We have used it as a tool to implement pre-processing of data and feature extraction logic. Python comes with a library function dedicated for machine learning algorithms which helped us in using our algorithms without any complications. Most machine learning algorithms cannot take text as direct inputs but has to be transformed into numerical values. Feature Extraction is the procedure where we convert our raw text data into usable form.

- NLTK feature extraction with bag of words
- TF-IDF term weighting

2.2.1 Converting Words to Features with bag-of-words

NLTK (Natural Language Tool-Kit) is a leading platform in Python to work with human language data. It provides easy to use interfaces to over 50 corpora and lexical resources such as wordnet, along with a suite of text processing libraries for classification, tokenizing, stemming, tagging, parsing, and semantic reasoning [11].

Table 1 - NLTK parts of speech tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>English Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ</td>
<td>adjective</td>
<td>new, good, high, special, big, local</td>
</tr>
<tr>
<td>ADV</td>
<td>adposition</td>
<td>on, of, at, with, by, into, under</td>
</tr>
<tr>
<td>CONJ</td>
<td>conjunction</td>
<td>and, or, but, if, while, although</td>
</tr>
<tr>
<td>DET</td>
<td>determiner, article</td>
<td>the, a, some, most, every, no, which</td>
</tr>
<tr>
<td>NOUN</td>
<td>noun</td>
<td>year, home, coast, time Africa</td>
</tr>
<tr>
<td>NUM</td>
<td>numeral</td>
<td>twenty-four, fourth, 1991, 14:24</td>
</tr>
<tr>
<td>PRP</td>
<td>particle</td>
<td>at, on, out, over, per, that, up, with</td>
</tr>
<tr>
<td>PRON</td>
<td>pronoun</td>
<td>he, their, her, its, my, I, us</td>
</tr>
<tr>
<td>VERB</td>
<td>verb</td>
<td>is, say, told, given, playing, would</td>
</tr>
<tr>
<td>*</td>
<td>punctuation marks</td>
<td>, ;</td>
</tr>
<tr>
<td>X</td>
<td>other</td>
<td>erset, esper, duvou, gr, univeristy</td>
</tr>
</tbody>
</table>

One of the powerful usages of NLTK is the parts of speech tagging. It tokenizes words and labels them as a noun, an adjective, a verb etc. Table 1 provides a small list of tags [7]. NLTK also has provided us with tools for chunking which means grouping words with similar meaning. This proves to be very effective in doing feature extraction for sentiment analysis as we can group adjectives and synonyms as per their occurrence in the text to be either positive or negative. With this tool, we can train our algorithm with words of type adjective in our dataset and associate those words with negative or positive categories. The logic can be very simple. Therefore, for the new document that we want to evaluate, the algorithm will look for similar words or synonyms, which it has been trained on, and classifies the new texts accordingly as shown in Figure 1, as a result of execution of the Python code below.

```python
import nltk
sentence = """The Graduation ceremony saw many emotional students and family"""

words = nltk.word_tokenize(sentence)
tags = nltk.pos_tag(words)

print(tags)
```

**OUTPUT:**

```
[('The', 'DT'), ('Graduation', 'NNP'), ('ceremony', 'NN'), ('saw', 'VBD'), ('many', 'JJ'), ('emotional', 'JJ'), ('students', 'NNS'), ('and', 'CC'), ('family', 'NN')]
```

Figure 1 - NLTK output tokens

2.2.2 Converting Words to Features using TF-IDF Vectorizer

In a large text corpus, some words are present which provides little or no meaning to the actual content of the document. TF-IDF (term frequency-inverse document frequency) creates a document-term matrix from the provided text data. It calculates term frequency-inverse document frequency value for each word and fills the matrix. TF-IDF is the product of two weights, the term frequency and inverse document frequency [14].

\[
TF-IDF = \text{term frequency} \times (1/\text{document frequency})
\]

(1)

Term frequency is a weight representing how often a word has occurred in a document. If the word occurs several times in a document, then we can say that it will increase. On the other hand, inverse document frequency is weight representing how common a word is across the document and if the occurrence is more the TF-IDF weight decreases. Converting a document into word vector format, counting number of word appearances and then using TF-IDF transform on the matrix gives us the TF-IDF Vectorizer. TF-IDF Vectorizer formula is manipulated a bit for utilization [14].

\[
idf(t) = \log \left( \frac{|D|}{\mid \{d : t \in d \} \}} \right)
\]

(2)

Here, \(d\) is the number of documents where the term \(t\) appears, and we are adding 1 to the formula to avoid division by zero. So, the formula for TF-IDF stands as [14]:

\[
tf-idf(t) = tf(t, d) \times idf(t)
\]

(3)

A high weight of the TF-IDF calculation is reached when we have a high term frequency (TF) in the given document (local parameter) and a low document frequency of the term in the whole collection (global parameter). To explain the concept, consider the code example below, and its result in Figure 2:
2.3 Learning Algorithms

Learning consists of gathering samples of data, extracting similar properties from them and then trying to predict properties of unknown data using different learning algorithms. Also, features are regarded as the multi-dimensional entry when each sample has more than one instance. Learning algorithms can be divided into two forms: supervised and unsupervised.

In supervised learning [14] the analysis guided by response variable (that is, data comes with additional attributes that we want to predict). For example, the sample belongs to two or more classes and we want to learn from already labeled data how to predict the classes of unlabeled data. The general idea is we will be provided by ‘n’ number of samples or data. Let us assume that they can be categorized into two types. The data sample will be run through desired algorithms which will try to find a pattern in the sample and associate it with the already provided label with each data. Afterward, when the training is complete and we will take unlabeled samples and feed in the algorithm which will try to label the new samples by matching previously learned features.

While in unsupervised learning [14] the analysis is not guided by a response variable (that is the input data consists of some vectors without any corresponding target values). The aim in this type of learning is to group similar featured data which can be denoted as clustering or to determine the distribution of data within the input space known as density estimation or to project data from higher dimension to two or three dimensions for the purpose of visualization.

We will be using Python library ‘sklearn’ which comes with pre-defined algorithms and training datasets. We have taken five of the classifiers in consideration and we will discuss in detail about them.

2.3.1 Naïve Bayes Classifier

Naïve Bayes classifier is a supervised learning approach [12]. This supervised classifier was given by Thomas Bayes and hence the name. According to this theory if there are two events, p1 and p2 then the conditional probability of occurrence of even p1 and p2 has already occurred is given by the following mathematical formula [3][4]:

$$P(p_1 | p_2) = \frac{P(p_1)p_2}{P(p_2)}$$

(4)

2.3.2 Bernoulli Naïve Bayes

In a Bernoulli Naïve Bayes or multivariate Bernoulli [11] event model, features are independent Booleans describing inputs.

In documents where binary terms are of more priority rather than term frequencies, this classifier proves to be efficient. If \(x_i\) is a Boolean expressing the occurrence or absence of the \(i^{th}\) term from the vocabulary, then the likelihood of the document given a class \(C_k\) is given by:

$$p(x | C_k) = \prod_{i=1}^{n} p_{x_i}^{x_i}(1 - p_{x_i})^{1-x_i}$$

(5)

Where \(p_{x_i}\) is the probability of the class \(C_k\) generating the term \(x_i\). Bernoulli model is popular in evaluating short texts which comes in handy for our research as tweets are generally small texts.

2.3.3 Multinomial Naïve Bayes Classifier

It applies Naïve Bayes algorithm for multinomial distributed data and it is also regarded as one of the best algorithms for text classification where the data are typically represented in word vector format [12][14]. The distribution is parameterized by vectors \(\beta_y = (\beta_{y1}, ..., \beta_{yn})\) for each type \(y\), where \(n\) is the number of features and \(\beta_{yi}\) is the probability \(P(x_i | y)\) of feature \(x_i\) appearing in a sample belonging to type \(y\). The relative frequency counting is:

$$\beta_{yi} = \frac{N_{yi} + \alpha}{N_y + \infty n}$$

(6)

Where, \(N_{yi} = \sum_{x \in T} x_i\), is the number of times feature \(i\) appears in a sample of type \(y\) in the training set \(T\), and \(N_y = \sum_{i=1}^{T} N_{yi}\) is the total count of all features for type \(y\) [3].

2.3.4 Support Vector Classifier

Support Vector classifier is a supervised machine learning algorithm which can be used for both classification and regression challenges. The classifier plots each data in an n-dimensional plane with a unique value of each feature and then classifies the data into classes by a hyperplane [15].
2.3.5 Logistic Regression Classifier

The multinomial logistic regression is used when classifying data into one of the two classes as considered in our study [6]. Most of the previous studies show this classifier to be better performing than other for discriminative classes of data.

3 Techniques employed for building Application

We have gathered a large dataset from people reviews with negative and positive emotions. We will do the experiment on the dataset first by feature extraction through NLTK (Natural Language Tool-Kit) and then we will try the same procedure with TF-IDF (term frequency–inverse document frequency) vector format. This is a general comparative study between the two types of text analysis feature extraction methods.

3.1 Sentiment Analysis with NLTK framework

This section discusses the basic sentiment analysis framework that is used to judge the emotions. The first step we applied is data collection, then preprocessing of data which included feature selection, next step being feeding the classifiers with the data we have and let it classify the text based on the feature we provided and finally display the result. The dataset is gathered from various data mining blogs and websites.

Our study depends on the unbiasedness of the data gathered and the test results will vary according to datasets.

3.1.1 Data Collection

Sentiment analysis can be done on any data [8]. The data can be either scrapped from the websites or can be collected from any data set already available with proper labeling on various data analysis websites. Scraping and extracting data from web with proper labeling is a tedious job and needs more time to create a lengthy database since more the data in hand, the more accurate is the prediction of our classifiers.

We have scrapped the data from various review websites and merged them with an already labeled dataset available online.

We believe that the basic language construct used for any or product reviews are similar when people express their feelings in Twitter as the general vocabulary remains the same. We have used first 10,000 texts to be our training set and the remaining 662 as our testing sample.

3.1.2 Pre-Processing of Data

First, we took the text data and appended positive or negative tags with each of the texts since the files for positive and negative was separate but not tagged. We then divided each sentence into tokens and corresponding parts of speech. Since the emotion of a sentence can be determined by the type of adjective or adverb used, we will take them as our primary feature in classifying texts. We have stored all the types of adjectives, found in the training dataset provided, in a dictionary and extracted a feature-set (a tuple in python terms) comprising of those adjectives and its adjacent tag, either positive or negative. The code below assigns each of the words which are adjectives (“J”) or adverb (“R”) with the corresponding sentiment value (either positive or negative).

Then the classifiers were trained on these features. The testing dataset is then fed to the classifiers without the labels and their prediction is compared with the data result that we have.

```python
featuresets = [(find_features(rev), category) for (rev, category) in documents]
```

Pre-processing of the data is an important part since here we are cleaning the data from any unnecessary noise, the extraction of adjectives are stemmed to their roots, also synonyms are considered so that any spelling mistake or use of a similar kind of word can be detected by the classifier. This data cleansing affects the accuracy of our classifier.

3.1.3 Classification

We divided the dataset into training and testing datasets, training being the first 10,000 texts.

```python
training_set = featuresets[:10000]
testing_set = featuresets[10000:]
```

We have considered 5 text analysis classifiers such as Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Logistic Regression, and Linear Support Vector Machine. All the classifiers have different parameters to play with for best accuracy on the dataset that we have worked on, but had the same methods for training the dataset and predicting the outcome of the testing data. For example:

```python
LogisticRegression_classifier.train (training_set)
print("LogisticRegression_classifier accuracy percent:", (nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)
```

As we can see the interface provided for training and testing data for the classifiers are quite simple.

The term Pickling is a process where a Python object is converted into a byte stream. Alternatively, we can say that pickling is similar to “Serialization” or “Marshalling” process. However, pickling has greater usage than marshalling as once a python object has been stored as pickle object it can be persisted in the database or transferred in a network. In our scenario once trained we saved the classifiers as “pickle” objects. We can also store the text data in tensor structure available in python instead of text files or we can also create the model based on No-SQL databases [5]. Since the training takes a considerable amount of time and we do not want to spare that amount of time whenever we run our application. Along with training the classifier we also test the accuracy of testing data set and print out the most informative feature it has received while training which helps it classify the data.
3.1.4 Display Results

On the first training and testing sets, we print out the accuracy for each classifier and its most informative feature. This gives us an idea of how all the classifiers are performing on our datasets and also the features that we found common with similar type of texts. This informative feature will vary with each dataset and so is the accuracy, since each classifiers works differently, while converting word to learning features.

As shown in Figure 3, The feature “flat” (as pointed by the arrow) is true and its associate tag which is “neg : pos” has a ratio of 14.2 to 1.0. This dataset contains more negative vocabulary since people use more words while describing a negative review than a positive review [1].

![Figure 3 - Naïve Bayes Classifier accuracy and the most informative features in the dataset](image)

As shown in Table 2, the accuracy of all the classifiers are pretty close to each other and more than 70%, and the Logistic Regression and Bernoulli Naïve Bayes having the highest accuracy. Also, the accuracy of the model increases with an increase in the amount of dataset [10].

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multinomial Navie Bayes</td>
<td>73.564</td>
</tr>
<tr>
<td>Bernoulli Navie Bayes</td>
<td>74.018</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>74.019</td>
</tr>
<tr>
<td>Linear Support Vector</td>
<td>72.205</td>
</tr>
</tbody>
</table>

3.2 Text Classification with TF-IDF feature extraction

As we have discussed above, TF-IDF Vectorize is the product of two weights, the term frequency, and inverse document frequency:

\[
TF-IDF = \text{term frequency} \times (1/\text{document frequency})
\]

The goal is to fit our data in the TF-IDF Vectorizer so that we can train our model.

3.2.1 Data manipulation

We have converted our text file into comma separated value format and to manipulate the data easily we are using a python library called “Pandas” [13]. It is a package providing fast, flexible and expressive data structures designed to make working with “Labelled” data easy and intuitive. The two primary data structures of Pandas are Series and DataFrame. In our scenario, we are building a DataFrame for the analysis. The DataFrame head is shown in Figure 4:

![Figure 4 - Data Frame](image)

We are getting help from another Python library called, “model_selection”, which helps us to split arrays or matrices into a random train and test subsets. The parameters in the train_test_split method of the library take in test size and train size of the data. It randomly shuffles the data also to keep training and testing set distinguishable and returns a list containing train–test split of inputs but we use Pandas methods to shuffle the data.

```python
#Dividing Dataset
X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33, random_state=53)
```

The above code splits the data into train and test modules where $X\_train$ and $X\_test$ contain all the text part and $y\_train$ and $y\_test$ contains all the label associated with it. Once the data is ready for fitting we convert it into vector form which is a crucial part of building the classifier model.

3.2.2 Transforming Data

The code below transforms the training and testing texts into vector matrix format giving decimal numbers to each of the features or words present in the document.

```python
tfidf_train = tfidf_vectorizer.fit_transform (X_train)
tfidf_test = tfidf_vectorizer.transform (test)
```

We manipulated the parameter “Stop Words” of TF-IDF. Stop words are words which are filtered out as they refer to the most common words in the language such as the, is, at etc. which provides no function towards the overall meaning of the sentence. Some phrases are also there which are regarded as stop words in English which only confuses the classifier while predicting the correct outcome. By assigning the parameter in TF-IDF to English it automatically removes those noises from our data.

3.2.3 Classification

The code below is used to provide our classifiers with the transformed data as an argument.

```python
mn_tfidf_clf = MultinomialNB (alpha=0.1)
mn_tfidf_clf.fit (tfidf_train, y_train)
pred = mn_tfidf_clf.predict (tfidf_test)
```

The code above uses our classifier with the transformed data as an argument.

```python
score = metrics.accuracy_score (y_test, pred)
print ("mn_tfidf_clf accuracy: %0.3f\%% score")
```
Here we have provided the labeled data as well as the parameter so that the classifiers can extract features and associate it with the data label. The classifier method for prediction, takes the testing dataset as the parameter and outputs, and this label is then compared against the \( y_{test} \) which we did not provide to the classifier. The \text{accuracy_score} \ method gives a percentage of times the classifier was correct with the pre-defined label of the testing dataset as shown in Figure 5. As shown in Figure 5, the graph is upwards, and the Y-axis is for the “True-Positive” classification, which means that the label was ‘positive’ and our classifier also predicted it as ‘positive’. Also, the accuracy will increase with an increase in data [10].

![Figure 5 - TF-IDF Classifier Comparison: Label = Positive](image)

The comparison of these classifiers suggests that Bernoulli classifier outperformed the other classifiers in predicting the correct outcome as shown in Table 3.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multinomial Navie Bayes</td>
<td>74.60</td>
</tr>
<tr>
<td>Linear Support Vector</td>
<td>74.90</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>74.90</td>
</tr>
<tr>
<td>Bernoulli Navie Bayes</td>
<td>76.40</td>
</tr>
</tbody>
</table>

3.2.4 Twitter API module

Twitter has given the developers a wonderful platform for text analysis since they allow parsing every tweet. The data received from the API is in Jason format from where we extracted the text and utilized it for sentiment analysis.

Twitter requires us to create a developer account doing which we get four different keys for accessing the API.

- Consumer ckey="XOXOXOXOXOXOXOXOXOXOX"
- Consumer secret="XOXOXOXOXODODODODODODODOCOCOCOCOC"
- Access token="XOXOXOXOHOOHOHOHOHOHOH"
- Access secret="AOAOAOAOAOAOAXOXOXOXOXOXOXOXOXOX"

The keys are changed for security purposes. We created a module where we load the saved classifiers. We then imported that module for the class listener which streams the tweets and forwards it to the classifier methods.

3.2.5 Voting System

We created a method which takes the output values from all the classifiers for each text and stores them into an array. We then use the statistical function \text{mode} \ to take the most common output among all the classifiers and take it as the final sentiment value. In the same way, we give a confidence percentage showing that our classifiers are in sync while giving the judgment.

\[
\text{choice} = \text{votes.count(mode(votes))} \\
\text{confidence} = \frac{\text{choice}}{\text{length (votes)}}
\]

The code above represents how confidence percentage is calculated. The statistical mode of the votes and the length of the votes are taken, and their ratio is returned to the calling function. The “choice” denotes the number of classifiers who voted as the statistical mode of all the votes. Taking the ratio of choice and the total number of votes, we generate the confidence percentage. If the confidence vote is below 60%, we consider the tweet to be neutral.

4 Results and Discussion

The Twitter API is a live streaming API which gives us access to approximately a million tweets per day and that much data is enough to analyze the sentiment of people on a specific topic. Figure 6 shows the filtered tweets from the “Jason” object with the keyword Hurricane Harvey. The sentiment values as positive or negative are underlined and written at the end of each tweet line.

![Figure 6 - Tweet classification on keyword Hurricane HARVEY](image)

The sentiment graph in Figure 7 is plotted with \( y = y + 1 \) if the sentiment is “positive” or, \( y = y - 0.4 \) if the output sentiment is “negative”. The increment of ‘1’ for “positive” and decrement of ‘-0.4’ for “negative” are chosen arbitrarily only for plotting this graph. As shown in Figure 7, it is evident that negative
tweets about Harvey are more which made the gradient of the graph fall towards the X-axis.

![Figure 7 - Sentiment Graph on keyword Hurricane HARVEY](image)

5 Conclusion

The study can be scaled to bigger proportions where it can be used for market analysis for a product launched currently by a company or for judging public sentiment on a product or on the result of different sports games. For this, a global mapping of data can be used to analyze which part of the country is having a more negative sentiment about a product and which part does not. This way a company can change its marketing strategy and re-think the way to launch a product.

In future, we can discard the “Bag-of-words” approach and consider the document as a whole. Considering the whole document while extracting features may give a better edge for understanding the general meaning of a document.

6 References


