Twitter Sentimental Analysis

Ashutosh Deoghare

Abstract:

In this paper I have addressed the problem of sentimental analysis on twitter dataset from Kaggle. I have used number of machine learning models with Tf-idf to classify whether the tweet is positive or negative. The most accurate model for classifying tweets correctly was Support Vector Machine.

1) Introduction:

Twitter is a popular social networking website where members create and interact with messages known as "tweets". This serves as a medium for individuals to express their thoughts or feelings about different subjects. Various parties such as consumers and marketers have done sentiment analysis on such tweets to gather insights into products or to conduct market analysis. Furthermore, with the recent advancements in machine learning algorithms, I am able to improve the accuracy of my sentiment analysis predictions. In this report, I am attempting to conduct sentiment analysis on "tweets" using various machine learning algorithms. I am classifying whether the tweet is positive or negative.

2) Methodology:

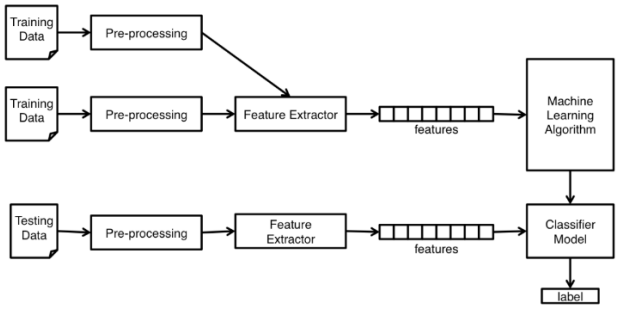

Fig. Schematic Diagram of Methodology

I have used different feature sets and machine learning classifiers to determine the best combination for sentiment analysis of twitter. I have also experimented with various pre-processing steps like - punctuations, stop words, twitter specific terms and lemmatization. I have investigated the following features - unigrams, bigrams, trigrams, compound of Unigrams and Bigram and compound of Unigrams, Bigrams and Trigrams. I have finally trained classifier using various machine-learning algorithms – Multinomial Naive Bayes, Logistic Regression and Support Vector Machine.

3) Data:

The dataset which I have used is from Kaggle.com and it basically belongs to Stanford University. To download this dataset simply click on this link: http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip

The dataset consists of 1.6 million user tweets with their username, id, date tweets posted, sentiment score and query. This dataset is already annotated by Stanford University students on the basis of emoticons like if the tweet contains smile emoticons then they have annotated it as positive tweet and if the tweet contains sad emoticon then they have annotated it has negative tweet. The proportion of positive and negative tweets in the dataset is equal that is 800000:800000 and they have been annotated as 4 for positive tweet and 0 for negative tweet.

4) Data Preprocessing:

As we are familiar with twitter, we know that the people use different kind of symbols, repeated words like @mentions, RT (retweet), emoticons, extra numbers, punctuations and many more to write a tweet and to do classification with this raw data was not possible. So, I have applied an extensive set of pre-processing steps to decrease the size of the feature set to make it suitable for learning algorithms.

3.1) HTML Encoding:

For some of the tweets the HTML encoding was converted to text, it was occurring in the dataset as '&', '&quot' etc. So, to get rid of such kind of text I had to decode HTML into general text using Beautiful Soup.

3.2) @mentions:

As we are familiar with twitter we know that we use @username to tag some user. This @username is of no use when it comes to sentimental analysis. So, I had to completely remove it using Regular Expression (r'@[A-Za-z0-9]+'**).**

3.3) URL links:

The third part for cleaning was to deal with URL's. I thought that even though it carries information, but I preferred to ignore that information and remove the whole URL. The URL was in two formats in the dataset i.e. www.xyz.com and http://www.xyz.com. So, to remove both kind of URL's I have used Regular Expression. (r'https?://[^]+') and (r'www.[^]+').

3.4) UTF-8 BOM:

The UTF-8 BOM is a sequence of bytes (EF BB BF) that allows the reader to identify a file as being encoded in UTF-8.

In my dataset I have found patterns of characters like "\ufffd" which I came to know that it was of UTF-8 BOM. I have replaced this kind of patterns by '?'.

3.5) Hashtags:

As we know hashtag in tweets can be very useful feature to get insights about the tweet. So, I thought that I will only get rid of '#' sign from the tweet keeping its following word intact. For this I have again used Regular Expression ([^a-zA-Z]) and replaced '#' with space.

3.6) Lemmatization:

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

In addition, with all these preprocessing and cleaning of data there was also other small stuff's which I have included in data cleaning for example I have converted words like isn't to is not, didn't to did not and so on.

5) Feature Selection and transformation:

For feature and some part of preprocessing I have used Tf-idf vectorizer from the scikit learn library of python. Now what id Tf-idf?

Tf-idf stands for Term Frequency-Inverse Document Frequency. It converts each tweet into document and generate sparse matrix of the words which are present in the tweets. The sparse matrix is the matrix in which most elements are zero, but it tells the frequency of each word present in the document.

Term frequency

It increases the weight of the terms (words) that occur more frequently in the document. Quite intuitive, right?? So it can be defined as tf(t,d) = F(t,d) where F(t,d) is number of occurrences of term 't' in document 'd'. But practically, it seems unlikely that thirty occurrences of a term in a document truly carry thirty times the significance of a single occurrence. So, to make it more pragmatic, tf is logarithmically scaled so that as the frequency of terms increases exponentially, we will be increasing the weights of terms in additive manner.

$$tf(t,d) = \log(F(t,d))$$

Inverse document frequency

It diminishes the weight of the terms that occur in all the documents of corpus and similarly increases the weight of the terms that occur in rare documents across the corpus. Basically, the rare keywords get special treatment and stop words/non-distinguishing words get punished. It is defined as:

$$idf(t,D) = \log(N/Nt \in d)$$

Here, 'N' is the total number of files in the corpus 'D' and 'Nt ∈ d' is number of files in which term 't' is present. By now, we can agree to the fact that tf is an intra-document factor which depends on individual document and idf is a per corpus factor which is constant for a corpus. Finally, tf–idf is calculated as:

$$tf\text{-}idf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

Now, Tf-idf was able to do some amazing things. As it has counted the word frequency of each word in the tweet it was also able to count the word frequency of N-grams in the tweet.

N-grams:

N-gram is a contiguous sequence of *n* items from a given sample of text or speech.

N-gram of size 1 is referred to as a "unigram", size 2 is a "bigram", size 3 is a "trigram" and so on.

So, I have used five different types N-grams and its combinations as features to get the best accuracy with the machine learning models.

Features which I have used for analysis are:

a) Unigrams
b) Bigrams
c) Trigrams
d) Unigrams + Bigrams
e) Unigrams + Bigrams + Trigrams

Along with this feature selection Tf-idf function also allows to remove stop words directly by passing just a parameter.

```
vect = TfidfVectorizer(analyzer = "word", ngram_range=(1,2), stop_words = 'english')
```

Fig. Tf-idf Vectorizer used in the project

Here, is the screenshot of Tf-idf vectorizer used in this project the ngram_range parameter allowed me to choose N-grams as my features like passing (1,1) will choose only Unigrams, (2,2) only Bigrams and (1,2) both Unigrams and Bigrams. Along with this the stop_words parameter allowed me to remove all the English stop words from the document when I pass 'english' to it.

So, for the experiment purpose I have used tweets with and without using stop words.

6) Machine Learning Models:

I have used three machine learning models for this project.

6.1) Multinomial Naïve Bayes:

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

6.2) Logistic Regression:

Logistic Regression also known as Maximum Entropy Classifier. It is a binary classifier and used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It can handle both dense and sparse input

6.3) Support Vector Machine:

These are the set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

The support vector machines support both dense and sparse sample vectors as input.

7) Results:

I have done 10-fold cross validation on training data every time when I have used different features. And, as I have experimented my models using both with and without using stop words.

The results are as follows:

Without Stop words:

Accuracies of 10-fold cross validation on training data:

	Multinomial Naïve Bayes	Logistic Regression	Support Vector Machine
Unigrams	76.09%	77.20%	76.49%
Bigrams	71.60%	71.56%	70.87%
Trigrams	58.58%	58.54%	58.40%
Uni + Bi	76.98%	78.36%	77.23%
Uni+ Bi + Trig	77.01%	78.25%	77.70%

Accuracies on testing data:

	Multinomial Naïve Bayes	Logistic Regression	Support Vector Machine
Unigrams	76.25%	77.34%	76.65%
Bigrams	71.81%	71.80%	71.07%
Trigrams	58.86%	58.83%	58.72%
Uni+ Bi	77.09%	78.52%	77.33%
Uni+ Bi+ Tri	77.12%	78.44%	77.84%

With Stop words:

Accuracies of 10-fold cross validation on training data:

	Multinomial Naïve Bayes	Logistic Regression	Support Vector Machine
Unigrams	77.58%	79.79%	79.22%
Bigrams	79.46%	79.10%	78.76%
Trigrams	72.99%	72.77%	73.23%
Uni+ Bi	79.46%	82.17%	81.59%
Uni+ Bi+ Tri	79.58%	82.19%	82.37%

Accuracies on testing data:

	Multinomial Naïve Bayes	Logistic Regression	Support Vector Machine
Unigrams	77.61%	79.92%	79.36%
Bigrams	77.98%	79.36%	78.88%
Trigrams	73.10%	73.46%	73.60%
Uni+ Bi	79.59%	82.41%	81.67%
Uni+ Bi+ Tri	79.69%	82.46%	82.57%

8) Comparison with Baseline Model:

Now, to compare my models with some of the baseline models earlier built by some researchers [1]. I have used only 10,000 tweets of the same dataset and have used only unigrams as the model features same as the researchers used in their model and measured the accuracy.

Comparing with them my models were performing very well and the accuracies of the same is given below.

	Baseline	My Project
Naïve Bayes	49.61%	73.85%
Logistic Regression	49.61%	75.80%
SVM	49.54%	73.70%

9) Conclusion:

With all my experiments I conclude that SVM performed best with 82.57% accuracy using features as Unigrams, Bigrams and Trigrams. Also, I want to add that models with Tf-idf predicted tweets accurately with stop words in the tweets.

10) Reference:

[1] A twitter sentimental analysis using NLTK and Machine Learning Techniques (International Journal of Emerging Research in Management &Technology ISSN: 2278-9359 (Volume-6, Issue-12))