Twitter Sentiment Analysis

CS 583 – Project 2

Ashwin Bansod (661344487)

Krutarth Joshi (670996389)

Fall 2016

Abstract

Twitter Sentiment Analysis is the quintessential task in the profound understanding of Text Classification and by extension, Natural Language Processing. There has always been a trend of modifying and fitting the general classification techniques from the fields of Image Recognition and likewise to accommodate in Text Classification. We've tried to do the same. The algorithms we've implemented include Support Vector Machines, Naïve Bayesian Classifier, Random Forest, k-nearest neighbors, and Deep Learning.

1. Introduction

Natural Language Processing is an extremely difficult task for Machine Learning. Modern NLP algorithms are based on machine learning, especially statistical machine learning. The paradigm of machine learning is different from that of most prior attempts at language processing. Prior implementations of language-processing tasks typically involved the direct hand coding of large sets of rules. The machine-learning paradigm calls instead for using general learning algorithms — often, although not always, grounded in statistical inference — to automatically learn such rules through the analysis of large corpora of typical real-world examples. A corpus is a set of documents (or sometimes, individual sentences) that have been hand-annotated with the correct values to be learned. We use Machine Learning to generate sentiment information from the tweet data.

2. Technique

The first and primary task of any data mining project is to collect and analyze the data. The real world data is generally incomplete, lacking attribute values, contains noisy data, error or outliers and in some cases inconsistent across attribute values. Specifically, text data is high unstructured and noisy in many cases. To achieve better insights and build better predictive models using efficient algorithms, it is necessary to clean data and extract meaningful information form it.

In this project, we are dealing with tweet data, which is one of the most unstructured and noisy text data. Tweets are an informal means of communication and contains spelling mistakes, bad grammar, usage of slangs, URLs, emoticons, elongated words (to stress on some point), hash-tags, mentions, html tags (if data is collected using some html parser), etc. [1]. The tweets are classified according to sentiment as positive, negative and neutral. The absence of positive words and presence of negative words in tweet classifies the tweet as negative, and vice-versa. Those tweets that do not clearly express the positive or negative sentiment is classified as neutral.

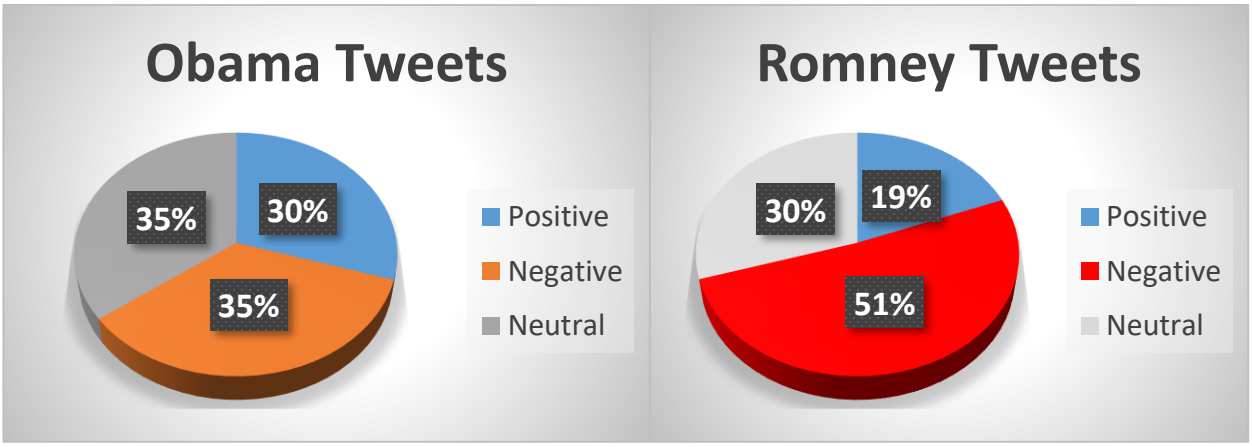
2.1. Data Cleaning

Data cleaning or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data [2]. Following are the data cleaning techniques that we tried and used.

- HTML escape characters: Using html.unescape we removed the html escape characters from the text.
- Decoding tweet: decoded the tweet to 'utf-8' format to remove any unwanted, unreadable characters.
- Tweet Preprocessor: Python provides a tweet preprocessor library which provides functionality to clean tweet text by configurable parameters to remove hash-tags, mentions, reserved words, URL's, emoji's and smileys. Using this tweet preprocessor library, we cleaned URL's, hashtag's, mention's, reserved words.
- Apostrophe Removal: Using regular expression we cleaned the commonly used apostrophe's in the text data.
- Punctuation's Removal: Using regular expressions we removed the punctuations in the tweet text.
- Shorten Elongated words: Using regular expressions we shortened the elongated words (for ex: happppppyyyyy, whatttttttt to hapyy and whatt).
- Single letter word: We removed the words with single letter or non-alpha numeric words from the tweet text.
- Stopwords: Earlier we removed the stopwrods from the tweet text. But, since the corpus of total unique words was very small, we decided to keep the stopwords in the corpus. This decision helped us in improving the overall accuracy by 1-1.5%.
- Stemming: Earlier we tried to use the porter stemmer to stem the words in the tweet text. But since the total words in corpus were very small, we decided the remove stemming and use the original form of words as features.

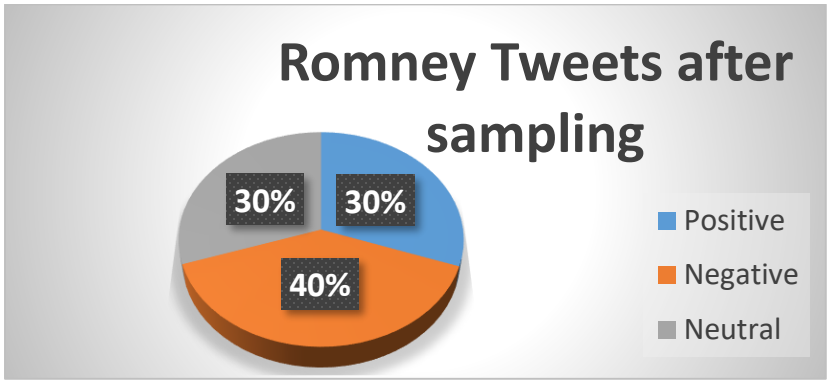
2.2. Data Sampling

Analyzing the distribution of data is very important for any data science project. Skewed and improper distribution of data can greatly impact the precision, recall and F-score of the model. When we analyzed the given training data for Obama dataset, we found that the distribution of positive, negative and neutral class was equivalent. But for Romney, the distribution of classes



was very skewed. More than half of the tweets in dataset were of negative class and positive class tweets comprised only 19% of entire data.

Therefore, using the Romney dataset without any data sampling would result in very high recall for negative tweets and very low recall for positive tweets, which is not a very good predictive model. In order to mitigate the effects of the improper distribution, we decided to use the over sampling by randomly duplicating the positive class tweets. Thus making the distribution of classes equivalent. Below is the distribution of Romney dataset after sampling.



We even tried under sampling, by reducing the number of negative class examples by random selection. However, this was not very effective as the total number of tweets was already very less. Further reducing the number of tweets resulted in under-fitting. The corpus of words was reduced significantly.

2.3. Feature Selection

We tried various feature selection algorithms in the project, namely count-vectorizer, hashing-cectorizer, TF-IDF vectorizer, Word2Vec, Doc2Vec.

- Count-Vectorizer: Count Vectorizer counts the occurrence of each word in corpus and selects the words with most occurrences as features. Using only count vectorizer was not giving very good results.
- Hash-Vectorizer: It converts the given text documents to a matrix of token occurrences. Using hash-vectorizer was not giving good results.
- TF-IDF: It converts the raw documents into a matrix of TF-IDF vectors. We used TF-IDF in association with the Count-Vectorizer to get the best results. The words with highest TF-IDF values were selected as features by the model.
- Word2Vec: Word2Vec was created at Google. It uses similarity between two words to compute feature vector. The feature vector for each word needs to be created by training it over the corpus. We tried it training it over the corpus of words present in the training data corpus. But the results were not so good as the number of words in the corpus was very less. So, we tried using a pre-trained word2vec model of tweet words which consist of 20 million words. Then we created the feature vectors using this model and used it for prediction. The results were comparable with the best results that we got. If we work more on this, we can further improve the performance using word2vec.

2.4. Classification Methods

Classification is the process where you train your model on the features extracted and then use that trained model to predict the test data. Overall accuracy of the model is important, but more important are the precision, recall and F-score of the positive and negative class. Even though we are doing three class classification, we are more concerned about the F-score of positive and negative class. We tried various classification techniques as below:

2.4.1. Naïve Bayes Classification

We used Multinomial Naïve Bayes classifier to train our model using the TF-IDF feature vector created in pre-processing step. Naïve Bayes classifier works very well in text classification because of its independence assumption. We considered one-gram and bi-gram features for the classification. Using the grid search, we found that the optimal value of alpha should be 0.099.

2.4.2. K-Nearest Neighbors Classifier

We tried using K-Nearest Neighbor classifier on this data, as we thought that it would work well with feature vector trained using word2vec. However, our assumption was wrong. The performance of K-Nearest Neighbor classifier was poor. Even after suing the optimal setting for the classifier, the performance was not improved.

2.4.3. Linear SVM Classifier

Support Vector Machines or SVM performs very well with text data and this tweet dataset was no exception. With proper data cleaning and TF-IDF feature selection algorithm, the overall performance of SVM was best for three class classification. We used grid search to find the optimal parameter values for the LinearSVM classifier.

2.4.4. Random Forest Classifier

We tried random forest classifier as it is a meta classifier that fits on decision trees for various sub-samples of dataset. The performance of random forest classifier was descent compared to the SVM. The overall accuracy was 2% less than that of SVM. The recall and F-score for positive class was less compared to SVM.

2.4.5. Voting Classifier

This is a strict hard voting classifier which gave majority voting among the above mentioned classifiers. Using the Voting classifier, the overall accuracy was improved by 1%.

2.4.6. Deep Learning

We tried using Deep Learning techniques on the given dataset by referring to [4] [5] [6] [7]. The Deep Learning network used the below architecture:

1) Skip gram method: Google Word2Vec

2) Representations of sentences and documents: The Paragraph vector is basically an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents.

3) Generate a vector representation of the whole sentence/paragraph.

4) Using a CNN to summarize documents.

The performance of three class classification using the Deep Learning was not the best, as hoped. The overall accuracy was 3-5% less than that of SVM classifier. But, for two class classification, the overall accuracy of Neural Networks was 8-10% more than that of SVM. The introduction of neutral class introduces a lot of noise in the data.

3. Evaluation

We used various evaluation measures to measure the accuracy and correctness of the model, also to check for possible overfitting of the data. We used k-fold cross validation on every model and computed the average accuracy of the model. The model would not be considered a good model if the difference between the minimum and maximum value of the k-fold cross validation is more than 4%. In order the deal with this issue, we did random shuffling of data to make the data as evenly distribute for each class. K-fold cross validation accuracies for Naïve Bayes and LinearSVM classifiers are 60% and 61% with a variation of about 2%.

On the given test dataset, the performance of SVM was the best followed by the Naïve Bayes classifier. The F-score and recall for negative and positive class on the Obama dataset was 61% and 56% resp. Whereas, for Romney dataset, the recall for negative class more, thus impacting the F-score value. The F-score for negative and positive class on the Romney dataset was 65% and 45% resp.

4. Conclusion

After completing the project and implementing various techniques to generate the model, we understood that, there is no single correct way to build the classifier. Each and every dataset is different and analysis of the data, to implement the proper data cleaning techniques along with appropriate algorithm, is very important. A lot of research is required to find the best suitable classification algorithm.

In this project we determined that SVM and Naïve Bayes works the best with the given tweet dataset. Sampling of improperly distributed dataset is very important in order to improve the recall of positive tweets. The overall accuracy for Obama dataset and Romney dataset are similar. Neural Networks performs very well in two class classification it performs 10% better than SVM. However, by introduction of neutral class, a lot of noise in inserted into the classifies and therefore the results are not as good as that of SVM in three class classification.

5. Bibliography

1. https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/
2. http://betterevaluation.org/sites/default/files/data_cleaning.pdf
3. http://stackoverflow.com/questions/30795944/how-can-a-sentence-or-a-document-be-converted-to-a-vector
4. Palangi, Hamid, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. "Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.4 (2016): 694-707. Web.
5. Tai, Kai Sheng, Richard Socher, and Christopher D. Manning. "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks." *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*(2015): n. pag. Web.
6. LeCun, Yann. "Text Understanding from Scratch." https://arxiv.org/pdf/1502.01710.pdf.
7. Misha Denil, Misha, and Nando De Freitas. "Modelling, Visualising and Summarising Documents with a Single Convolutional Neural Network." (n.d.): n. pag. Web.