Twitter Sentiment Analysis

Manmohan Dogra *Neel Mehta*

Abstract

Twitter has become a central site where people express their opinions and views on political parties and candidates. Emerging events or news are often followed almost instantly by a burst in Twitter volume, providing a unique opportunity to gauge the relationship between expressed public sentiment and electoral affairs. One of the NLP issues that have received the greatest attention is sentiment analysis. It entails examining the provided data and making predictions about its sentiment. Opinion mining, sentiment mining, and subjectivity analysis are just a few of its numerous titles. There are several uses for it, including chatbots, product reviews, and aspect reviews. Sentiment analysis comes in a variety of forms, including fine-grained sentiment analysis, aspect-based sentiment analysis, document-level sentiment analysis, and sentence-level sentiment analysis. Sentiment analysis at the sentence level is the main focus of this project. While traditional content analysis takes days or weeks to complete, the system demonstrated here analyzes sentiment in the entire Twitter traffic about the election, delivering results instantly and continuously. It offers the public, the media, politicians, and scholars a new and timely perspective on the dynamics of the electoral process and public opinion.

Introduction

We're provided with a labeled dataset consisting of tweets from the 2012 election; the class labels are the sentiments denoted as (-1,0,1). The problem statement is to build a machine learning model that can help classify the sentiment of the tweets 2012 election. Our approach to this problem is very logical first we preprocess the data to make it clean and ready for machine learning models to learn from it. Followed by training the preprocessed data on multiple sentiment models such as linear classifiers, LSTMs, and Bert transformers. We compare the results from all the models using a combination of preprocessing steps using the metric of precision, recall, and F-score. We're able to achieve the best performance using the SVM (RBF kernel) with an F-score of 0.63.

Techniques

1) Preprocessing (Naive Bayes, SVC, Decision Trees, Bagging, XGBoost, AdaBoost):
Data cleaning and feature extraction were performed in the preprocessing stages. In the training stage, several deep-learning models were used. Detailed results are presented in the next section. The main objective of our study is to evaluate deep learning models. We used k-fold cross-validation with k = 10 to determine the performance of the algorithms. All of them were tested with word embedding and TF-IDF.
Text cleaning is a preprocessing step that removes words or other components that do not contain relevant information, and thus may reduce the effectiveness of sentiment analysis. Text or sentence data include white space, punctuation, and stop words. Text cleaning has several steps for sentence normalization. All datasets were cleaned using the following steps:

- Cleaning the Twitter RTs, @, #, and the links from the sentences;
- Stemming/lemmatization;
- Converting the text to lower case;
- Cleaning all the non-letter characters, including numbers;
- Removing English stop words and punctuation;
- Eliminating extra white spaces;
- Decoding HTML to general text.

A certain processing method was then performed depending on the dataset to facilitate model formation. For the given dataset, we dropped the columns that are not useful for sentiment analysis purposes: {“unnamed0”, “unnamed5”, “date”, “time”} and converted class label values {-1, 0, 1} to {0, 1, 2} (0 = positive, 1 = neutral, 2 = negative).

After the datasets were cleaned, sentences were split into individual words, which were returned to their base form by stemming. At this point, sentences were converted into vectors of continuous real numbers (also known as feature vectors) by using two methods: word embedding and TF-IDF. Both kinds of feature vectors were the inputs for the deep learning algorithms evaluated in the study.

2) Preprocessing(LSTM, BERT-transformers)

- For the big models, we first tokenize and then numerical the texts correctly.
- The difficulty here is that each pre-trained model, which we will fine-tune, requires exactly the same specific pre-process-tokenization & numericalization-as the pre-process used during the pre-train part.
- Additionally, as we are not using RNN, we have to limit the sequence length to the model input size.
- We add special tokens at the start and end of the sentences as most of the models require special tokens placed at the beginning and end of the sequences.

3) Sentiment model

The algorithms we tried were Naive Bayes, Random Forest, AdaBoost, SVM, Bagging, XG-Boost, LSTM, and BERT-transformers.

- **Naive Bayesian classifier:** As our first approach, we chose Naive Bayes to be our first algorithm to try on. It is a fast and no iterations algorithm since the probabilities can be directly computed. If the conditional Independence assumption holds, it could yield good results.

- **Random Forest:** The next approach was to train a random forest algorithm as it prevents overfitting by using multiple trees and would be a good option for this amount of data. To improve it, we tried to increase the iterators and tested boosting methods.

- **Adaboost:** Next, we used AdaBoost to boost the performance of the classifier, and it is less prone to overfitting as the input parameters are not jointly optimized.

- **Support Vector Classifier**: The reason we used an SVC as it is effective in high dimensional spaces as we’ve high dimensional space after using tfv_metric to train our data. It uses a subset of training points in the decision function (called support vectors), which is memory efficient. To improve its performance, we used the “RBF” kernel, with increased estimators.

- **XGBoost**: We used this technique as it boosts the classifier performance by providing a more direct route to the minimum error, converging more quickly with fewer steps, and simplifying calculations to improve speed and lower compute costs.

- **Bagging:** Bagging offers the advantage of allowing many weak learners to combine efforts to outdo a single strong learner. It also helps in the reduction of variance, hence eliminating the overfitting of models in the procedure.

- **LSTM**: After working with other machine learning models, we shifted towards LSTMs, as they are particularly designed to have a long-term memory that is capable of understanding the overall context better than other neural networks for NLP problems. The results were decent, not the best we'd say as any neural network needs tons of data to train and learn the context. To increase its performance we fine-tuned the parameters and selected the AdamW optimizer well known for its decay.

- **BERT-transformers:** We had to try transformers, even though our dataset is really small to confidently train a transformer, but BERT is very well known for its capacity to learn textual context and build many reliable text models due to its attention to masking. BERT practices to predict missing words in the text, and because it analyzes every sentence with no specific direction, it does a better job at understanding the meaning of homonyms than previous NLP methodologies, such as embedding methods. To improve its performance we fine-tuned the parameters and selected an AdamW optimizer well known for its decay. Surprisingly, the performance of BERT was close to what SVC produced, but SVC yielded more balanced results compared to BERT. We assume it happened as a transformer, it'll need a lot more data to perform better.

Experiment results

Below is the compilation of results obtained using 10-fold cross-validation by training different models

		precision	**recall**	**f1-score**
	Negative	0.58	0.62	0.6
	Neutral	0.54	0.45	0.49
Naive Bayes	Positive	0.59	0.64	0.62
		precision	**recall**	**f1-score**
	Negative	0.57	0.54	0.55
	Neutral	0.49	0.52	0.51
Decision Tree	Positive	0.55	0.56	0.56
		precision	**recall**	**f1-score**
	Negative	**0.59**	**0.63**	**0.61**
	Neutral	**0.54**	**0.59**	**0.53**
SVC(RBF)	Positive	**0.61**	**0.59**	**0.6**
		precision	**recall**	**f1-score**
	Negative	0.58	0.64	0.61
	Neutral	0.53	0.49	0.51
Bagging	Positive	0.61	0.59	0.6

		precision	recall	f1-score
	Negative	0.57	0.54	0.55
	Neutral	0.49	0.52	0.51
AdaBoost	Positive	0.55	0.56	0.56
		precision	**recall**	**f1-score**
	Negative	0.52	0.54	0.53
	Neutral	0.51	0.39	0.44
XGBoost	Positive	0.52	0.61	0.56
		precision	**recall**	**f1-score**
	Negative	0.6	0.56	0.43
	Neutral	0.48	0.4	0.35
LSTM	Positive	0.57	0.59	0.74
		precision	**recall**	**f1-score**
	Negative	0.48	0.52	0.45
	Neutral	0.54	0.39	0.27
BERT	Positive	0.55	0.59	0.67

Conclusion

We presented a system for Twitter sentiment analysis of tweets by Obama and Romney. Our data processing infrastructure and statistical sentiment model evaluate public sentiment changes in response to emerging political events and news as they unfold. From our experiments, we compare the results from all the models using a combination of preprocessing steps using the metric of precision, recall, and F-score. We're able to achieve the best performance using the SVM (RBF kernel) with an F1-score of 0.61. The architecture and method are generic and can be easily adapted and extended to other domains.

We've learned that there are more logical preprocessing can be done, by using masking and attention model for the transformers. More embedding models can be tested out, to generate an accurate representation of the text which can help models learn more efficiently, and perform better. More amount of data can be used to train large models to perform better than the above models. It is very difficult to analyze sentiments, as it is hard for any model to learn parts of sentiments, such as entity, aspect, and opinions which can be as granular as we go. To fully understand the sentiments, a vast labeled corpus of parts of sentiments would be needed to be trained on a large model to get promising results.