Prediction of Congress Party Based on Tweets Using NLP

Haixi Zhang, Yiwei Han
University of Rochester, USA
hzh104@u.rochester.edu, yhan32@u.rochester.edu

***Abstract* — Based on a set of train data of tweets, the number of favorable, retweets, and year, using the test data set to predict which political party a particular tweet belongs to. Use methods including natural language processing, machine learning, etc.**

***Keywords* — Prediction, Accuracy, Natural Language Processing, Cross-Validation, Party, Tweets**

I. Introduction

In this Kaggle Competition, we are asked to use a training data set to develop a model that can correctly predict the party of each tweet owner. After implementing the prediction model developed by using the training data set, we are expected to be able to predict the party of each tweet with relatively high accuracy. Given the initial full-text data, we did a lot of data clean steps to make the data meaningful. Also, including setting ngram equals to (1,1) and (1,2) to vectorize the text. And, we employ various models to predict and choose the one which has the highest accuracy, which is Logistic regression. And the final accuracy checked by Kaggle is 89.49%%, which is well enough to predict the party of the owner from a given text.

II. Descriptive ANALYSIS

A. Congressional_tweet_training_data.csv

This data set is used for training the model, thus the model being trained can be used for prediction on the test data set, and report accuracy. This data set consists of 592,803 data (tweets). And each tweet has 6 features, including Id, favorite_count, full_text, hashtags, retweet_count, and year. Those are the features we need to select and train. The label is party_id, which we use to predict.

B. Congressional_tweet_test_data.csv

This data set is used for prediction, which means to see how well our model works. It consists of 265,000 tweets. The features are the same as the training data set. After we developed the model, we will apply the model to this data set to do prediction and report accuracy.

C. Sample_submission_data.csv

This dataset consists of 265,000 tweets. It is the concise version of the test data set and only has Id and party features. It is being used to calculate and compare the accuracy of the party we predicted and the ones given by the test set.

D. Visualization

First, we visualize the proportion of each party in the training data set. See Figure 1. We found out there are more Democratic than Republicans who post tweets. The difference is about 10%.

Second, we visualize the length distribution of tweets in each part. See Figure 2. We found out that for both parties, most tweets' lengths are between 100~150 words. However, Democrats generally have longer tweets, since the bimodal distribution shows there are more tweets between 180~250 words, which does not exist among Republicans. As a result, we can see that there's a different tweets length pattern for each party.

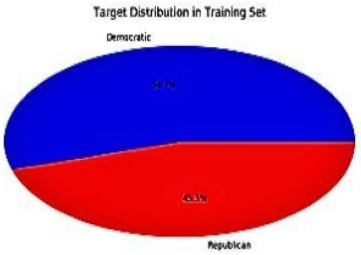

Fig. 1. Percentage of tweets in each party

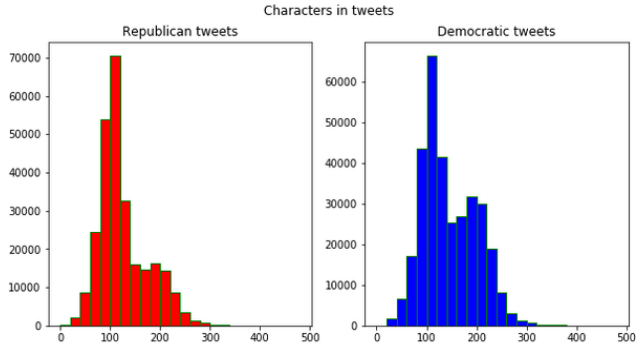

Fig. 2. Length distribution of tweets for each party

Third, we make the word cloud plots on the train data ['full_text'] column and specify the party. And obviously, we can notice that there are a lot of "HTTPS" words, which is meaningful in our research. Therefore in the data cleaning step, we should clean this word.

Besides, we can notice that both parties focus on the small businesses part and the American people. While the Democratic are more concerned about health care, house pass, high school… And the Republican is more concerned about public health and proud jobs… Hence the two party's tweet content does has something different.

Fig. 3. Word Cloud of Democratic tweets

Fig. 4. Word Cloud of Republican tweets

III. Classification Methods

A. Data Preprocessing

First of all, we do a lot of data cleaning steps, especially on the [‘full_text’] and [‘hashtags’] columns. In the beginning, we do data cleaning on both columns, while the accuracy is lower than not changing the [‘hashtags’]. Hence, we choose to only change the [‘full_text’] as clean the starting word ‘b’ of every tweet, lower all the texts; remove the punctuations and not-used characters(‘&, xa9, x80’...); replace the typical contractions (such as changing ain’t to am not…). And we take advantage of the nltk package, from where we import the Stopwords and Tokenize tools. The reduced texts, both the train dataset and the test dataset, are stored in the [‘Text_Cleaned’] column. After that, we do the label encoder on the party column, change “R” to 0 and “D” to 1. Then store the value in the new column, [‘P’].

B. Set the Model

To prepare for the further model fit, we choose to use two different methods, which are the CountVectorize and Tf-Idf methods to create the sparse matrix of numerical vectors. By experimenting with different sets, such as ngram_range=(1,2), min_df=2, we finally decided to use ngram_range(1,1) and min_df=2. Under this set, the accuracy and the needed time are both optimized. And by empirical, we decided to combine both [‘Text_cleaned’] and [‘hashtags’] together for a model to train on. And by several trials, we discovered that: [‘Text_cleaned’] + 2* [‘hashtags’] can produce the best accuracy.

C. Vectorizer

Since CountVectorizer and TfidfVectorizer both are capable of converting the text data into vectors, which are sparse matrices that can be used for further analysis. And CountVectorizer mainly focuses on counting the number of times a word appears in the documents so that the CV will produce the results based on the most frequent words. In this case, the rare words will be neglected and decrease the performance. Therefore, the TfidfVectorizer will be employed, which is capable of taking the overall document weightage of each word and will probably perform better than CV.

In this project, the ngram part larger than 2 will be very hard to calculate since this is a such huge dataset. Therefore, use the CountVectorizer and TfidfVectorizer with ngram(1,1) along with TV(1,2) (on LogisticCV to produce the best accuracy) to produce the sparse matrix of Words Vectors.

D. Logistic Regression Cross-Validation

The first method we thought of is simply using Logistic. regression and employ the ‘lbfgs’ or ‘saga’ solver to better deal with the large dataset. We choose to use the ‘lbfgs’ solver, set cross-validation to be 2, rand=265, and max_itear=200 of the training dataset to train the model.

E. Linear Ridge Classifier

Then we also tried to use Linera_model.RidgeClassifier(), which is a method that, compared to Logistic regression, uses a different way to do regularization. And Instead of using the cross-entropy function, the Ridge() uses the MES + L2 penalty as the cost function instead. However, the accuracy is not improving.

F. Multinomial Naive Bayes Model

Then the Naive Bayes method comes to our mind, which is generally used where there are discrete features (for example – word counts in a text classification problem). And the good news is that this method is extremely fast, while the accuracy is still lower than the logistic model.

IV. classification Results

At first, we thought this project was very similar to the previous Problem Set 4. So we realize that the main point of this project is doing data cleaning. At first, we simply start with the simple characters cleaning and lemmatizing. But it is really slow. We wasted a lot of time on this.

In the beginning, we used logistic regression, which is slow when we choose cv=5. So we changed too many different methods, including Naive Bayes, Ridge Classifier(), and SVM. But unfortunately, these models all produce around 55% accuracy.

After we did more training, we realized that the cleaning steps are the most significant. As a result, we add a lot of cleaning functions, such as characters, punctuations, constructions, URLs, emojis….. And we also notice that we can combine clean_text with hashtags to improve performance. Even further, we try to do data cleaning on the 'hashtags', but this causes the accuracy to decrease(the first row of Table.1, the 2nd column is with initial hashtags, and the 3rd column is with the cleaned hashtags). Therefore, we decided to not do data cleaning on the 'hashtags' anymore.

the best training data can be calculated by combining the cleaned texts with hashtags together (to be more specific, multiply the hashtag by 2 because this variable is more significant in analyzing). Then we used CountVectorizer and TfidfVectorizer to calculate the sparse matrix of the Word Vector and use that as the data for further training. Since in previous homework, the ngram(1,2) and (1,3) cannot perform better than (1,1), so we simply use (1,1) instead.

For the three models we choose, we found out that using TfIdf vectorizer along with the Logistic Regression Cross-Validation produces the best prediction accuracy, which is 89.03% (the highest). And the other models all produce lower accuracy.

Generally, processing the whole program at first takes a long time, but we fixed it so that the processing time gets quicker, and the model gets easier and clearer.

Models	Accuracy (with raw hashtag)	(with cleaned hashtags)
Logistic RegressionCV	89.012%	87.5%
+RidgeClassifier( )	88.175%	\
Multinomial Naive Bayes	87.119%	\

Table11. Under CV vectorize and the Accuracy

Models	Accuracy	(with cleaned hashtags)
Logistic Regression	89.032%(TV1_1) 89.492%(TV1_2)	\
RidgeClassifier()	88.940%	\
Multinomial Naive Bayes	87.505%	\

Table12. Under TV vectorize and the Accuracy

V. Conclusion

In this project, by carefully doing data cleaning, applying TF-IDF vectorizer, and using several predict models, the final model, which is the one with TF-IDF (ngram =(1,2)) and using Logistic Regression (cv=2,min=200), can eventually produce a prediction of 89.492% accuracy on the test data. In all, our model is very well working on using Congressional politicians' tweets to predict the party of the owner.