Tweets Political Ideology Classification

Han Cao hcao29@wisc.edu
Siyi He she83@wisc.edu
Qiwen Zeng qzeng36@wisc.edu

Abstract

Some reports[5] has been indicating that social media has a strong impact on political ideologies of the society. Emerging conspiracy theories suggest that other countries have been using social media platforms to manipulate presidential election of the United States. However, in order to massively monitor such behavior online involving technologies that enable computer to understand the political ideologies behind the texts. We proposed using machine learning algorithms to solve such a problem. We experimented Support Vector Machine, XGBoost, and Multinomial Naive Bayes on a data set collected from twitter. We eventually achieved an accuracy of 76%.

1. Introduction

1.1. Background

According to several articles released recently[5], Internet companies can potentially have the power of manipulating the political ideologies of the public to achieve the goals of private parties. We want to develop a tool that can help us understand the public opinions of political ideologies on social media.

Machine learning can be used to achieve such a purpose. Natural language processing has been significantly developed in the past decades. There already exist applications using machine learning to perform analysis on the sentiment of twitters.

We propose to use machine learning techniques to identify political ideologies behind social media texts. Given the context of the 2020 presidential election is happening, we want to classify whether a text implies Democratic or Republican ideas.

Twitter is one of the most influential social platforms. Many politicians or political organizations use Twitter to broadcast to their audiences. Thus we decide to train our model using labeled twitter posts.

The goal of this project is to train one or several machine learning models to classify political ideologies, specifically Democratic and Republican ideas, using machine learning models trained with Twitter posts.

1.2. Motivation

1.2.1 Related Event

The main motivation for researching this topic comes from the ongoing presidential election. We are curious about the trends of supporting Democrat or Republican on popular social media. By developing such a tool, we can monitor public opinions on social media. On the one hand, by doing so, we can potentially try to form a trend of public opinion shifting of political ideologies. By drawing connections with key events, we can investigate the impact of such events on public opinions. On the other hand, by using such a tool, we can also classify the user's political affiliation based on the text they posted.

1.2.2 Applications

If we eventually developed a decent model, we can potentially apply it to the tweets sampled from a larger data set to monitor the public opinions. Moreover, we may use that data to predict the result of the 2020 presidential election. Some other potential usages including:

1. Identify Political Bots on Twitter.

2. By adding some twitter users information, we may form a demographic map of political ideology.

2. Related Work

In work from 2009[10], Yu, Kaufmann, Diermeier established a pipeline for developing ideology classifier for political speeches. They deployed Naive Bayes model on speech data, and achieved F1-score of 63%. This paper provided us some insights on data-preprocessing and model ideas.

Here is another project[1], also try to classify political ideologies but with political speeches. In the paper, the authors discussed the performance of different models. We will borrow some experiences from this research to select models that are suitable for our problem.

3. Proposed Method

3.1. Support Vector Machine

The first model we present here is support vector machine. Support vector machine is a supervised learning algorithm. The algorithm classified given data by constructing one or multiple hyperplanes in a high dimensional space. There are multiple types of support vector machine, in this project, we use support vector machine with linear and rbf kernel. In the following, we will shortly introduce the principle of support vector machine. To fit a linear support vector machine, it is necessary to find two hyperplanes that are as far apart as possible and a training example must lie on each hyperplane. Figure 1 shows an example of linear support vector machine[9].

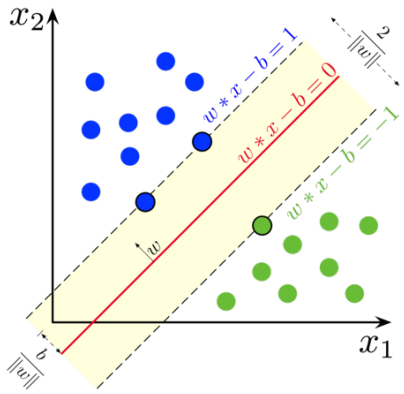


Figure 1. SVM example: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Radial basis function kernel (RBF kernel) is a popular kernel function used in various kernelized learning algorithms, which is commonly used in support vector machine classification.

$$K(x, x') = \exp(-\frac{\left\|x - x'\right\|^2}{2\sigma^2}) \tag{1}$$

$\left\|x - x'\right\|^2$ may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter[8].

3.2. Multinomial Naive Bayes

The second model we presented here is multinomial naive bayes. Naive bayes is also a supervised learning algorithm. It is a model that commonly used in text classification tasks. Hereby we introduce the principle behinds the algorithm. In multinomial naive bayes[7], the likelihood of observing a histogram x is given by

$$p(x|C_k) = \frac{(\Sigma_i x_i)!}{\Pi_i x_i!}\Pi_i {p_{ki}}^{x_i} \tag{2}$$

The multinomial naïve bayes classifier becomes a linear classifier when expressed in log-space.

$$\begin{aligned}\log p(C_k|x) \propto \log p(C_k)\Pi_{i=1}^n {p_{ki}}^{x_i} \\ = \log p(C_k) + \Sigma_{i=1}^n x_i * log p_{ki} \\ = b + w^T x\end{aligned} \tag{3}$$

where b=$\log p(C_k) and w_{ki} = \log p_{ki}$.

3.3. XGBoost

The last model we want to introduce here is XGBoost. XGBoost is a widely-used supervised learning algorithm as well. It is a decision-tree-based ensemble algorithm with a gradient boosting framework. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction[2]. When adding the new models, it will minimize the loss by using gradient descent algorithm. Based on this, XGBoost has a regularized objective for better generalization, and additive solution for generic objective function.

For XGBoost[3], the object function is

$$Obj = -\frac{1}{2}\Sigma_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{4}$$

This object function include the training loss and the complexity of trees. The optimal leaf weight is

$$w_j^* = \frac{G_j}{H_j + \lambda} \tag{5}$$

4. Experiments

In this section, we will show you how we did analysis on our raw data set and how we did data processing to make them more effective when applying our machine learning methods.

4.1. Dataset

The data set we use is from Kaggle [6]. Twitter Users were evenly selected from Republicans and Democrats. About 200 tweets were collected from each user. Overall, our dataset contains 84,502 tweets with party labels and the labels are roughly balanced.

Party	Handle	Tweet
Republican	222	42434
Democrat	211	42068
Total	433	84502

Table 1. This shows the raw data structure.

Table 1 shows the raw data structure.

4.2. Prepossessing Steps

Step 1. Remove Unrelated Characters

There are some characters that have nothing to do with our classification, such as the twitter handles, punctuation and digits. Figure 2 illustrates the parts we removed in step 1.

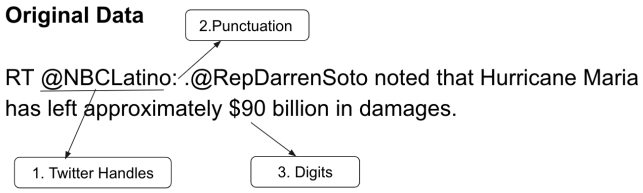


Figure 2. One example tweet from our data set.

Step 2. Tokenization Process and Remove Stop Words

In this step, we first tokenized strings into separate words. Then, by analyzing the frequency of each words, we found that the words with highest frequencies do not have any meaning and appear very often in both party's tweets.

word	frequency
the	37324
to	28725
of	15877
and	15340
in	13931
a	12087
for	11741
rt	9986
on	9093
is	7476
Republican's Tweets	

word	frequency
the	33980
to	28825
of	15483
and	14704
a	13076
in	12074
for	11382
rt	9068
is	7651
on	7642
Democrat's Tweets	

Table 2. These two tables show the 10 words with highest frequencies in republican's and democrat's tweets respectively.

Table 2 compares the 10 words with highest frequencies in republican's and democrat's tweets. We can observe that top 10 words for both party's tweets are the same with slightly difference in the order. All of those high frequency words are just stop words with no meanings.

In Figure 3 and Figure 4, we made random sample over the party's data respectively. This time we can observe some high frequency words with meanings such as "great" and "today". However, it is clear to see that the words with highest frequencies are still those uninformative stop words, such as "to" or "of". We removed those stop words using the stop word library.

One thing to notice, there is another stop word that is not in our usual stop words library here, which is "RT". It appears very often as the beginning string of our collected tweets (in the left side of both Figure 3 and Figure 4). Thus, we removed "RT" as the last part of step 2.

Figure 3. A word cloud based on random sampled word frequency of republican party's tweets

Figure 4. A word cloud based on random sampled word frequency of democrat party's tweets

Step 3. Stemming Process

By stemming the tokenized words, we made our data more dense and reduce the dimension.

Here, Table 3 displays the output based on the previous steps. The data output of this step are considered clean.

Steps	Output
Raw data	Today, Senate Dems vote to #SaveTheInternet.
Remove unrelated characters	Today Senate Dems vote to SaveTheInternet
Tokenization	[Today, Senate, Dems, vote, to, SaveTheInternet]
Remove stop words	[Today, Senate, Dems, vote, SaveTheInternet]
Stemming	[today, senate, dem, vote, savetheinternet]

Table 3. This shows the output of each steps.

Step 4. Join back the Tokenized Words

In order to treat every word as a feature, we want to vectorize them. However, the vectorization function TfidVectorizer we want to use requires corpus as input. Thus, in this step, we join back our tokenized words.

Here is an example. Suppose we have only limited word space ([today, tax, student, senate, trump, dem, vote, biden, savetheinternet]), and we use the previous cleaned tweet in table 3 as corpus: "today senate dem vote savetheinternet".

The following table 4 illustrate how words are processed into features:

word space/features	value
today	1
tax	0
student	0
senate	1
trump	0
dem	1
vote	1
biden	0
savetheinternet	1

Table 4. This shows the value of one tweet entry toward the whole word space.

4.3. Software, Tools and Resources

Data Source

- Our data set Democrat Vs Republican Tweets was retrieved from Kaggle[6]

Data Preparation

- The basic software we used for preliminary data analysis is R Studio.
- The word clouds were generated on `https://www.wordclouds.com` with the customized word frequency lists.

Machine Learning

- Jupyter Notebook

Version Control

- Github was used for version control of the code.
- Overleaf and Google Doc were used for documentation purpose.

4.4. Hardware

Some machine learning methods we used took a very long time to run. In this case, some of the procedures were done separately on laptops that have the higher performance.

Machine models:

- Macbook Pro 15" 2018
- Dell XPS13

5. Results and Discussion

5.1. Model Training and Hyper-parameter Tuning

5.1.1 Multinomial Naive Bayes

For the multinomial Naive Bayes model.We ran a grid search on the parameter alpha. The result is showing in the following graph.

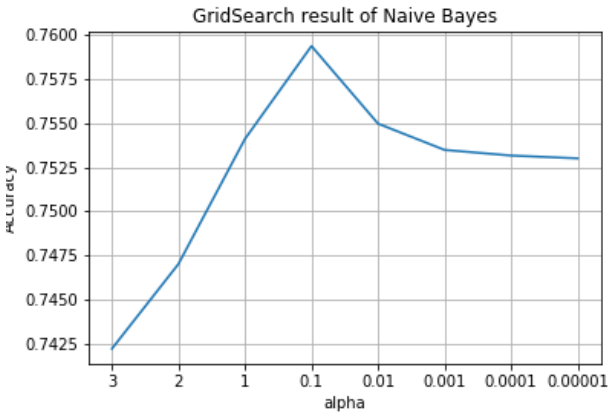


Figure 5. The Grid Search Result for Naive Bayes

The grid search result shows that when $alpha = 0.1$, our model performs the best. Therefore, we choose the model with $alpha = 0.1$ The accuracy of 4-fold cross validation is 75.94% eventually.

5.1.2 Support Vector Machine

According to this post (KOWALCZYK et al., 2020) [4], linear kernel support vector machine works the best for text classification tasks. Therefore, we chose linear kernel for our tasks. If linear kernel was chosen, then the only hyper-parameter that has significant impact to our model is C. To find the best hyper-parameters for our model, we ran a grid search on the parameter C. The result is showing in the following graph.

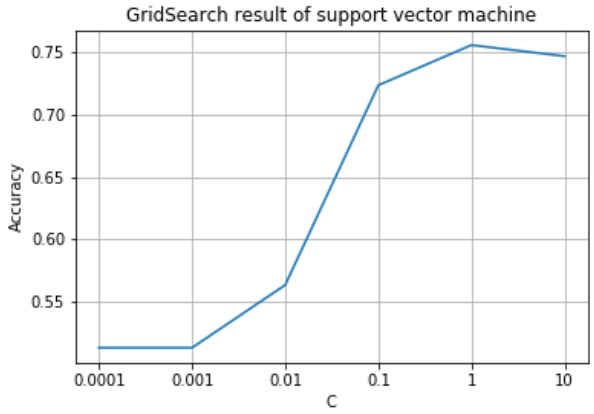


Figure 6. The Grid Search Result for Support Vector Machine

The grid search result on C shows that when $C = 1$ our model performs the best, therefore, we choose $C = 1$ for our final model for support vector machine. The accuracy of 4-fold cross validation is 75.94% eventually.

5.1.3 XGBoost

For XGBoost, we have more hyper-parameters to test than the previous two models. Consider that the training time cost is expensive, we decided to ran a random search on the hyper-parameters follows the set up as below.

Hyper-Parameters	Value Range
max-depth	[5,6,7,8,9,10]
min-child-weight	[0-10]
eta	[0-2]

The result of the random search shows as below:

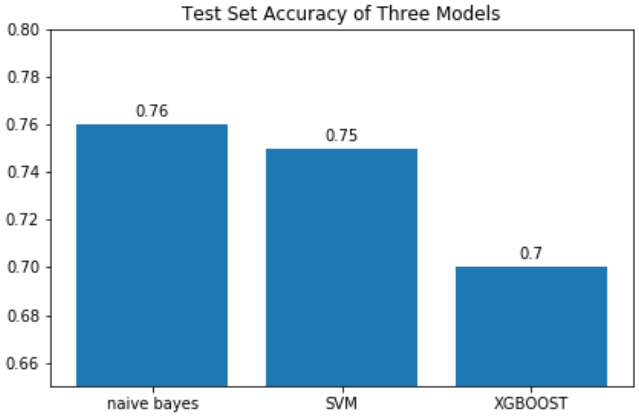
$\max_d epth$	min-child-weight	eta	accuracy
7	2.27	1.39	0.7035
8	4.91	1.1	0.707
6	6.8	1.5	0.695
6	1.2	4.0	0.698
6	4.3	1.06	0.698
5	5.9	1.2	0.691
8	1.2	1.7	0.71
9	4.2	1.03	0.694
6	7.4	0.59	0.695
6	6.5	0.71	0.694

In general we can see a pattern a here that higher max depth tends to yield model with high accuracy. Min child weight around 1 tends to improve the model accuracy. Based off our observation, we trained our final with the XGBoost model with the following hyper-parameters.

Max-Depth	Min-Weight	Eta
10	4.27	1.06

5.2. Model Evaluation and Selection

After we improved the performance of each model by hyper-parameter tuning, we present the test results for each model with the final selected hyper-parameters. The result shows in the figure below:



The result shows that support vector machine and Naive Bayes perform much better than XGBoost model. However, we don't see a large difference between the performance of Naive Bayes and support vector machine, although Naive Bayes performs slightly better than the Support Vector Machine. We want to further understand which model to choose.

5.2.1 McNemar's Test on Naive Bayes and Support Vector Machine

Due to the similar performance, we want to understand if the Naive Bayes model is the same as the support vector

machine model. Then we run a McNemar's test between these two models.

The confusion matrix of Naive Bayes model is shown as below:

		Support Vector Machine		
		Positive	Negative	Total
Naive Bayes	Positive	6758	1222	7980
	Negative	1056	8256	9312
	Total	9814	9748	17292

After we ran the McNemar's test over two model's prediction, we have p-value of 0.001. Therefore, we reject the null hypopthesis which states that the two model are the same. We conclude that the two models are fundamentally different.

5.2.2 Metrics

We want to further investigate what are the differences between the two models. We use confusion matrix to learn more about the pattern of the two models. We assume that 'Republican' is labeled as positive, and vice versa.

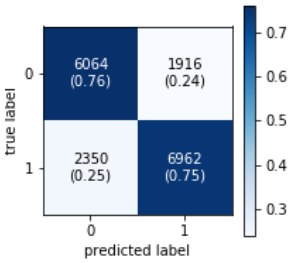


Figure 7. Confusion Matrix of svm

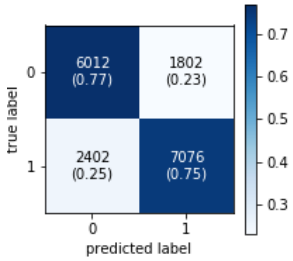


Figure 8. Confusion Matrix of Naive Bayes

The confusion matrix didn't reveal us any significant differences on traits among two models. Therefore, we try to use Accuracy, Precision, Recall, F1, and MCC metrics to further investigate the traits of the models.

Metric	Naive Bayes	SVM
Accuracy	0.757	0.753
Precision	0.747	0.748
Recall	0.797	0.784
F1	0.771	0.765
MCC	0.514	0.506

From the above table, we can see that Naive Bayes has noticeable higher recall rate, F1 score, and MCC. Since we assigned "Repulican" as our model, higher recall rate and F1-score can be interpreted as Naive Bayes model has better performance when identifying Republican tweets than the Support Vector Machine model. This feature can potentially be useful when performing certain types of task targeting republican tweets. The higher MCC also indicates that our model has a strong positive correlation to the test set. The above information told us that our model, in general, is better at predicting Republican tweets than Democratic tweets. Between two models, there is no significant difference among their prediction patterns.

Besides, Naive Bayes is way more faster to train than support vector machine when the number of features is huge.

6. Conclusions

After experimenting three types of models on performing political text classification, we conclude that Naive Bayes will be our final model. Naive Bayes has the highest accuracy of 76% on our test data set, and the high recall rate of Naive Bayes model allow us to do some potential future works based on such a model. Besides, Naive Bayes model also has lower training and predicting cost, which is potentially useful in industrial applications.

7. Acknowledgements

Thanks to Kyle Pastor who contributed to the initial data set on Kaggle[6]. We also appreciate the discussion posts about that data set, which inspired us a lot.

8. Contributions

Computational Task

- Han is responsible for data prepossessing, researching and implementing a machine learning pipeline
- Siyi is responsible for developing and tuning the models
- Qiwen is responsible for evaluating the model

Visualization Task

- Han is responsible for machine learning methods.
- Siyi is responsible for data set statistics.
- Qiwen is responsible for the rest.

Documentation Task

- Han is responsible for methods of the model, results and conclusion
- Qiwen is responsible for the introduction and related work
- Siyi is responsible for the experiment and the rest

References

[1] M. Bhand, D. Robinson, and C. Sathi.

[2] J. Brownlee. A gentle introduction to xgboost for applied machine learning, August 2016. Last Updated on April 22, 2020.

[3] T. Chen. Xgboost: A scalable tree boosting system, June 2016.

[4] V. KOWALCZYK, Murthy, Karuna, T. Khattak, and T. S. Nchabeleng. svm linear kernel good text classification, Jul 2020.

[5] M. Parks. Social media usage is at an all-time high. that could mean a nightmare for democracy, May 2020.

[6] K. Pastor. Democrat vs. republican tweets: 200 tweets of dems and reps, 2018. Last updated 27 April 2018.

[7] Wikipedia. Naive bayes classifier.

[8] Wikipedia. Radial basis function kernel.

[9] Wikipedia. Support vector machine.

[10] B. Yu, S. Kaufmann, and D. Diermeier.