478 Group Project: Predicting Twitter Authors

Chris West, Andrew Kimball, Jared Hanson
Department of Computer Science
Brigham Young University
cjwest4942@gmail.com, jaredthanson@gmail.com, andrewckimball@gmail.com

Abstract

Several models were trained to recognize twitter authors. Our initial goal was to find the feasibility of using machine learning models to predict authors of tweets and determine a good model for the task. We chose an initial data set that would emphasize recognizing author style over simply recognizing topics. We did this by looking at the most recent 500 tweets from all 100 US Senators, as they would write on similar subjects. We created three data sets with different levels of feature creation and reduction that resulted in varying degrees of accuracy.

1 Introduction

Twitter is a social media platform that allows users to share short messages with each other. Each user tends to have their own style of writing and tends to focus on certain topics. We were interested in seeing if this "style" could be learned by a machine such that it could identify novel tweets from United States Senators.

We did not expect language in tweets to make this problem completely separable due to the ambiguity of the English language. Different authors can write in similar fashions using similar words and may even quote each other. As such, we expected the problem to be difficult to solve with high levels of accuracy, but we did expect to make significant improvements over baseline accuracy as the authors tend to have a somewhat unique style of writing.

2 Methods

We discuss our methods of data collection and model selection and training in this section.

2.1 Data Source

We needed to select a subset of twitter authors to form our data set. Since we would be looking at word frequency, we wanted a selection of authors that would be writing on similar subjects, as to allow smaller subsets of feature words to contain more information. We chose US senators since they normally have a large history of tweets, a list of their twitter handles is easily obtainable, and they all generally tweet on similar subjects.

Twitter provides an API that can be used by students to obtain tweet data. After applying for access through a twitter developers account, we could query the twitter API for tweets. We looped through each Senator and retrieved the data[1] (In JSON form) for the 500 most recent tweets they had uploaded. We then could easily extract this data to form each of the different data sets used in our models.

We initially tried a bag of words approach to solve this problem. After some testing, we were not completely satisfied with the results, and we found ways to improve the feature set, which led to the creation of the following two data sets.

The data sets we created can be grouped into three categories: *Custom Bag of Words*, *Bag of words - TF-IDF Feature Engineering* and *tweet data coupled with text hashing using Latent Dirichlet Allocation (LDA)*. Each data set is explained below.

Custom Bag of Words

We created 60 features, each representing the 60 most commonly used words across the entire pool of tweets. We made sure all 60 words were significant by removing stop words[2]. We hypothesized that limiting the data set to 60 features would be beneficial to simplify the models. We also feared that the more features we added, the more empty and insignificant the data.

Each entry in the data set would represent one tweet, with each value representing the total number of times that particular word was referenced in the tweet. The output label was the Author of the tweet. It should be noted that most values in the data set are 0's. This is because for a given tweet, most of the words in this list of 60 most common words were not used at all. It should also be noted that, due to space constraints, this table shows just 7 of the 60 feature columns in the data set, nor does it show the label.

[1] data included the text of the tweet in all data sets described below. In some data sets it also include tweet meta data such as upload time or number of retweets and shares

[2] common words eg. "the" or "and")

Feature:	0	1	2	3	4	5	6
amp:	0	0	1	0	0	0	0
help	0	0	0	0	0	0	0
American	0	0	0	0	0	0	0
Senate	0	0	0	0	0	0	0
need	0	0	0	0	0	0	0
work	0	0	0	0	0	0	1
get	1	0	0	0	0	0	0
must	1	0	0	0	0	0	0
im	3	0	0	0	0	1	0
bill	1	0	0	0	0	0	0
support	1	0	0	0	0	0	0
make	0	0	0	0	0	1	0
people	0	0	1	0	0	1	0

Bag of words - TF-IDF Feature Engineering

In this data set, we relied on Scikit Learn's TfidfVectorizer class to help us create a more optimized bag of words approach. TF-IDF (or term frequency-inverse document frequency) is a statistic that reflects how important a word is to a document in a data set. It is calculated as the following:

$$\frac{num\_t\_appearances}{num\_terms \times log_{10} \frac{num\_terms\_in\_doc}{num\_docs\_with\_t}}$$

To create a matrix of TF-IDF features, we ran the text of all 48,625 tweets in our initial data scrape through Scikit Learn's TfidfVectorizer class. In addition to changing the way the features were calculated, we decided to include all the default hyper parameters, resulting in this new transformed data set including over 75,000 unique words, rather than 60 (as was the case in our custom approach). The success of this method (see **Final Results**) disproves our initial hypothesis that the data set should be relatively small.

Tweet data coupled with text hashing using LDA

This data set differed from the others in that we experimented with using other tweet metadata—$\{retweets,\ likes,\ replies,\ quotes,\ time\ of\ day\}$—in addition to the raw text and whatever features we could extrapolate from the raw text. We extrapolated features from the raw text using the Latent Dirichlet allocation algorithm. This would allow us to make all the the text into a small numeric feature set (6-12 features representing all the text). These features shall be referred to as topics.

Latent Dirichlet Allocation is a topic modeling algorithm that extracts the high-level topics from a set of documents. By using the LDA model for feature prepossessing, each tweet can then be given a score that represents its significance across n topics. These 6-12 scores are used as additional features in the data set

We used a prebuilt model from the Gensim Library for LDA text processing to create our topic groups. We created and tested several variations of data sets, each with a different number of topic features (6 - 15) to search for the best. Below is an example row from our data set:

Feature;	0	1	2	3	4
retweets	7	8	3	37	7
likes	9	27	16	99	21
replies	8	13	7	43	19
quotes	1	4	0	4	2
time_day	17	17	23	23	23
topic_1	0.00	0.00	0.00	0.00	0.00
topic_2	0.144	0.100	0.00	0.00	0.078
topic_3	0.225	0.000	0.438	0.000	0.000
topic_4	0.075	0.000	0.141	0.302	0.000
topic_5	0.079	0.000	0.000	0.067	0.000
topic_6	0.000	0.077	0.000	0.446	0.062
author_name	sen.S	sen.S	sen.S	sen.S	sen.S

2.2 Selected Models

We used different models for each of our 3 different data sets. On both the *Custom Bag of Words* and the *TF-IDF Bag of Words* data set, we used a multi-layer Perceptron Classifier, KNN Classifer, Support Vector Classifier, Decision Tree, Random Forest, Multinomial Naïve Bayes, and Complement Naïve Bayes.

On the *Tweet Data coupled with LDA* dataset we used the same models as the previous two data sets, but we excluded the Multinomial Naïve Bayes and the Complement Naïve Bayes models

3 Initial Results

Our first data set was comprised of the word counts of the 30 most common words in the senators' tweets. We assembled a data set from the 500 most recent tweets of each senator. We first tried training an MLP on this data set. We had thought that this model would do particularly well as the data set is made up of numerical data; however, this yielded a poor accuracy of 5.8%. 5.8% accuracy is nevertheless significantly higher than the baseline accuracy of 1%[3]. However, this did show that the problem was tractable and provided a starting point to make improvements from.

4 Data and Feature Improvements

After the poor results from our initial test, we decided to take a variety of different approaches in an attempt to improve the accuracy. These approaches fall under three categories, each category named after its data set.

Below are our attempts and improvements for each approach:

Custom Bag of Words

After the initial results from the MLP, we did not immediately give up on the data set. Our first attempt at improvement came by way of adjusting the hyper parameters using a grid-search. No matter the parameter, the results were practically the same. Our initial accuracy was just too low to have any hyper-parameter impact it in a meaningful way.

We also tried to increase the number of features to 60. This improved the overall results by over 1%.

[3]Baseline accuracy is calculated using the probability of an accurate guess when picking a senator at random

After failing to see results from the hyper-parameters or feature set size, we moved on to different ML models. Below are the best results for each model tried on a data set of 60 features.

Model	Accuracy
MLP	6.96%
K-Nearest Neighbor	5.44%
SVC	8.15%
Decision Tree	5.27%
Random Forest	5.48%
Naïve Bayes Complement	6.00%
Naïve Bayes Multinomial	6.70%

As it is pretty clear from the results above, we got relatively similar results, no matter the model, parameter, or data set size. However, 5 - 8% on this dataset is not as horrible a result as it may seem. Since there are 100 senators or 100 output nodes, an 8% accuracy is an 800% gain over the baseline of 1%.

After all this testing, we came to the conclusion that improving our accuracy above its current level would be impossible with the data set we created. Most likely this is due to the amount of empty values in each entry. At most, a senator might use a couple of the most common words in each tweet. We believed that this was due to the fact that every entry in the data would be comprised of 28 - 30 zero values. No matter the model, there was just not enough data. We later discovered that we were exactly incorrect in this assumption, that that it was not the zero values that was hurting our accuracy, but the small number of overall features.

We still wanted to find a way to improve our overall accuracy, so we decided to move on to different data sets entirely.

Bag of words - TF-IDF Feature Engineering

As discussed in Section 2, our second data set was created by doing a TF-IDF Vectorization [2]. We decided to try this approach because our background research indicated that this was an effective way to engineer features for Naive Bayes models and various other classifiers.

We initially chose to limit the data set to 60 features so that we could accurately compare the results with our custom data set. The results were better across all the models. We also ran a test with the default parameters of Scikit Learn's TfidfVectorizer class, which resulted in 70,000 features. The results were significantly higher than anything we had seen in our previous models (See Final Results).

The following table is a comparison of our Custom bag of words dataset with the TF-IDF. Both were run with 60 total features

Model	Acc. Custom	Acc. TF-IDF
MLP	6.96%	9.70%
KNN	5.44 %	6.10%
SVC	8.15%	10.35%
Decision Tree	5.27%	5.18%
Random Forest	5.48%	6.75%
NB Complement	6.00%	7.10%
NB Multinomial	6.70%	8.49%

Tweet Data coupled with text hashing using LDA

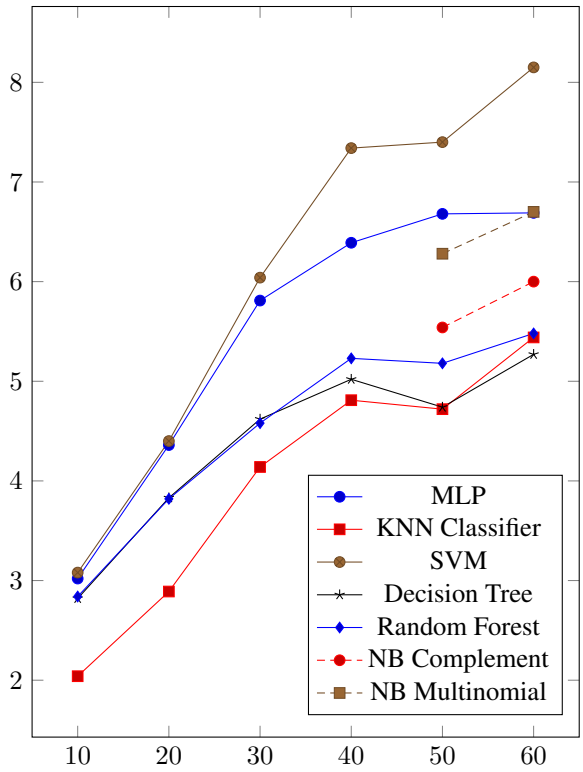
We also created a data set that includes topics generated using LDA, as discussed in Section 2. We found that using 12 topics, dropping "time of day", and doing no normalization yielded the best results. After dropping "time of day", we included the following tweet metadata: $\{retweets,\ likes,\ replies,\ quotes,\}$. We then tried passing this data set into several models. The results of testing can be found in the **Final Results** section.

5 Final Results

Custom Bag of Words

The chart below shows the accuracies of several different models used on the Custom Bag of Words data set.

Word Frequency Model Improvement over Baseline



The SVM model did distinctly better than all other models with an accuracy 8.15 times better than baseline when given a feature set of size 60. It appears that given more features, SVM may do even better; however, we found that other models were able to have significantly better results.

Bag of words - TF-IDF Feature Engineering

The table below shows the accuracies achieved by using the output of Scikit Learn's TfidfVectorizer class as the new data set. By default, the model includes as a feature every word in the entire data set (meaning all 48,625 tweets) minus stop words. The results of this feature engineering show a huge increase in accuracy. The MLP performed 54 times better than the baseline accuracy. It should also be noted that although the MLP model had better accuracy than the SVC and the two

Naïve Bayes models did, the MLP model was extremely slow in processing a data set with so many features.

Model	TF-IDF - 70,000 Features
MLP	54.40%
KNN	8.70 %
SVC	49.70%
Decision Tree	10.40%
Random Forest	31.84%
NB Complement	47.23%
NB Multinomial	44.15%

Tweet Data coupled with text hashing using LDA

By adding topics to the data set generated by LDA we were able to improve the feature set and improve accuracy. The three charts below show the improvement over baseline of the different models on data sets with 6, 10 and 12 topics. Each column shows the accuracy with other permutations to the data set such as whether or not the data was normalized[4] and whether or not the time of posting column was dropped from the data set.

6 Topics

	Normalized	No Norm	Drop Time
MLP	11.16	13.82	14.57
KNN	3.02	11.21	11.14
SVC	3.23	4.60	4.51
Decision Tree	8.39	9.19	9.00
Random Forest	11.19	11.00	10.62

10 Topics

	Normalized	No Norm	Drop Time
MLP	12.04	13.77	14.45
KNN	3.18	11.03	10.80
SVC	3.71	4.60	4.79
Decision Tree	8.73	8.13	8.64
Random Forest	11.07	10.75	10.68

12 Topics

	Normalized	No Norm	Drop Time
MLP	12.68	15.33	14.05
KNN	3.21	11.71	11.30
SVC	4.25	4.48	4.87
Decision Tree	8.73	8.76	8.64
Random Forest	10.89	10.22	10.07

6 Conclusions

We found that the MLP with the TF-IDF Vectorized data set was the best at classifying the tweets. This confirmed our initial assumption that an MLP model would perform the best at this problem, but we did not anticipate how much feature engineering would be required to achieve these results. We were proven wrong that the most optimal data set would have a minimal amount of features. Indeed, our best results came from including 70,000 features (representing every word found in all 48,625 tweets, excluding stop words).

As noted in the previous section, the downside of the MLP was that it was very slow to train. However, the Naive Bayes Multinomial and Complement and the Support Vector Classifier models ran very quickly and performed nearly as well.

We also found that relying solely on text yielded better results than using each tweet's metadata $\{retweets,\ likes,\ replies,\ quotes,\ time\ of\ day\}$ along with several topics from the LDA model. Although the Tweet Data coupled with text hashing using LDA performed better than our initial Custom Bag of Words, it was not nearly as accurate as the TF-IDF Vectorized approach.

In doing testing, we found that Weighted Complement Naïve Bayes performed better than Multinomial Naïve Bayes when using the full data set. This supports research done by others that found that Weighted Complement Naïve Bayes performed better at text classification problems than Multinomial [1]. Our results showed a 3% improvement over Multinomial Naïve Bayes when the full feature set was included. Though we did not measure time durations precisely, we found Naïve Bayes models trained faster than any of the more accurate models. This means that Naïve Bayes provides a way to trade accuracy for speed and that Complement Weight Naïve Bayes is the best model to use for said trade off.

7 Future Work

There are a couple tweaks and tests that could possibly generate better outcomes on our data. We only tried a range of topics between 6 and 12 and we could expand that out to see if there are better numbers of topics. Given that we were looking at classifying between 100 different classes, we it is not unlikely that adding topics will provide additional useful information to the models. Additionally, there are several other Naïve Bayes models provided by SciKit Learn that we could use such as such as a Gaussian Naïve Bayes. We could perhaps find a model that better fits the data we are providing as each NB model provided by SciKit Learn specializes on different kinds of data distributions. We could also try creating an ensemble of a few of these approaches.

Even though the MLP on the TF-IDF bag of words was the most accurate, we actually never got to run the MLP to completion. After running it for about 2 hours (150 iterations) we called it quits. It would be interesting to see how the accuracy would improve with a full run, however, we would need approximately 50 hours of compute time to accomplish this.

We also considered how one could modify this work for actual business application. Predicting the author of a tweet is not useful in itself, however, the strength of these models lie in their ability to process and find patterns in text with relation to an output class. For example, we could potentially alter this to predict the number of retweets or mentions a specific tweet will receive. This would have huge business applications, feasibly predicting the success of a post or comment before it was posted.

References

[1] Jaime Teevan Jason D. M. Rennie, Lawrence Shih and David R. Karger. Tackling the poor assumptions of naive bayes text classifiers. 2003.

[4] Min/max normalization was used when data was normalized

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.