Singularity Take-home Project: Text Document Classifier

Qiaoling Ye

August 14, 2020

1 Introduction

The goal of this project is to construct a text document classifier using training data, then test the performance on testing data. The given input data consists of 11,083 training documents and 7,761 testing documents. Both of them are labelled with 20 categories. Documents belong to the same category are saved under the folder named after the category.

Code reference. The raw data is under the folder: */text-document-classifer/00. Raw Data.*

2 Methodology

In this section, we will create an initial data set that, perform exploratory data analysis, construct features, train and test some predictive models.

2.1 Initial Data Set Generation

Since the original documents cannot fit in model training, we convert them to the following structure. Using this format, each row represent information from a document. The column *Category* consists of 20 different classes and column *Label* indicates either a document is from the training or testing data set.

File Name	Content	Category	Label
document 1 name	document 1 content	document 1 category	train/test
...	...	...	...

Code reference. This data set generation is performed in R using package `readtext`. R script and training and testing data are saved in the folder */text-document-classifier/01. Data Generation.*

2.2 Exploratory Data Analysis

Usually, exploratory data analysis (EDA) is to analyze both predictor and response variables; however, due to the nature of natural language, there is not that much strong coherence among predictors when they are ready to fit models. Therefore, we can perform EDA *on the training data* before feature engineering (Section 2.3), in order to visualize the distribution of response variable.

The primary concern is that either the training data set is balanced or not, i.e., if the number of documents are approximately equal or not among classes. Imbalanced data can make the prediction results biased by under-considering the minority class. Therefore, we check the number of documents per class and draw the box plot in Figure 1, which has five outliers. We perform oversampling/undersampling on the outlier classes to make sure they reach the lower/upper limit of the box plot, and compare the sample size per class before and after resampling in Figure 2.

Another concern is the length of the document: different length is correspond to different word frequency and thus may affect our TF-IDF score (Section 2.3.3) and cause result biased. There are some exceedingly long documents in the raw data (Figure 3 top pannel), so we remove the documents that has length longer than the 95th percentile, and we can see that the length distribution across categories is similar after truncation (Figure 3 bottom pannel).

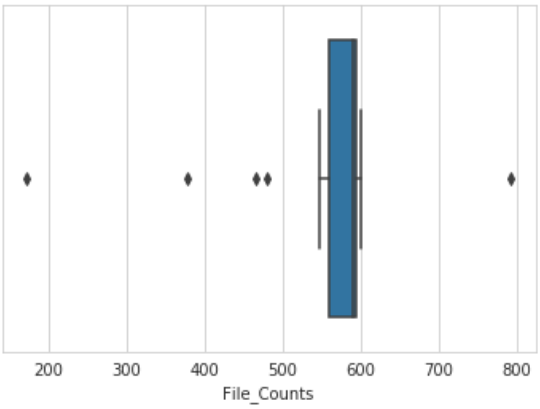

Figure 1: Box plot of the file counts per category in the training data.

Code reference. Please refer to */text-document-classifier/02. Exploratory Data Analysis* for details.

2.3 Feature Engineering

In this section, we will discuss how to clean, normalize and represent the data. The procedure in this section is performed over both training and testing data set.

2.3.1 Data Cleaning

The raw data consists of some redundant characters that have no prediction power, so we will remove them. Before removing these characters, we convert the documents to lower cases,

assuming upper and lower cases have the same meaning. Then we remove following redundant characters:

1. Special characters while reading the text document, such as `\n`
2. Punctuation signs such as `.,:"/@%`
3. Numbers
4. Stop words serving as connection words such as `too, from, its`
5. Possessive pronounce and other words that are common documents but no prediction power in this data set, such as `subject, re, et al`

After cleaning, the content in the document is a document of meaningful words that can be used to predict the category.

2.3.2 Data Normalization: Stemming and Lemmatization

Since one word can have different forms, we can recover the base form by lemmatization or generate new words using stemming. Stemming is a more aggressive method in model prediction (as it generate new words). We have more than 10,000 documents and thus we can reply on *lemmatization only* to modify words in the data set.

2.3.3 Data Representation

For the response variable, we convert the category to numerical numbers by mapping them to $\{0, 1, \ldots, 19\}$. For the simplicity, we won't list the exact mapping in the report. For the predictors, there are several methods to create features from text:

1. Word Count Vector. We can count the frequency of each words in the document, where words are from the corpus.
2. TF-IDF Vector. It is short for **T**erm **F**requency - **I**nverse **D**ocument **F**requency. to present the relative importance of the words in this document and the entire corpus.
3. Word Embeddings. The position of the word is learned from the text based the word itself and the words around it.

There are other methods, and we won't list them here. In this project, we used TF-IDF for our feature engineering because it is a simple but fast and effective strategy to yield good results in text document classification. The idea of TF-IDF is based on word frequency to convert text to vectors, without taking account the order or the sequence of the words. Let $TF_i(x)$ be the frequency of word x in a document i, and $N(x)$ be the total count in the data

set. We can calculate the inverse document frequency, $TDF(x) = \log\left(\frac{N+1}{N(x)+1}\right) + 1$. Then the TF-IDF score is defined as $TF(x) \times IDF(x)$, which calculates the importance of a word in one document relative to the corpus. We can use the complete word set as the corpus in this project.

We can also tune parameters in this process, including the number of grams (i.e., the number of tokens), maximum and minimum document frequency (DF) and maximum feature numbers. The following table showes the parameters in this project:

n-gram range	max DF	min DF	max features
{1, 2}	100%	10%	500

It is possible to further tune these parameters, but due to time constraint, I used these parameters for simplicity.

Code reference. Please refer to */text-document-classifier/03. Feature Engineering* for the details of data cleaning, normalization and representation.

2.4 Predictive Model

After constructing our features, we can train predictive models using the training data. Taking both time and data size into consideration, I choose classical machine learning methods rather than deep learning models. Let me explain the models that I used briefly.

Random Forest. It is a bagging method that aggregates the results from multiple decision trees using re-sampled features. Random forest in general can generate moderately good results for categorical data.

Multinomial Logit Model. It is in the generalized linear model family, and used (multinomial) logit function as a link function. Since we have 500 features, linear combination of predictors should be enough to predict the category.

Naive Bayes Model. It assumes independency between features and applies the Bayes rule to maximize the marginal probability. Since this assumption is usually assumed to be true in text document, it is a useful method in document classification. Another big advantage is that it runs pretty fast compared with other complex methods.

K-nearest Neighbor Model. It is non-parametric model by taking account the K-nearest neighbors. We have more than 10,000 data in the training set, it will take time to compute the distances.

Most of these models (except naive Bayes) need to tune hyper parameters, and we applied two rounds of cross validation to choose tuning parameters. The first round is a random search, which can search over a large range of parameters (along with cross validation). Then we zoom in the neighbors of the selected parameters and perform a more fine-grid search (along with cross validation). The scoring metric is "accuracy"/ "f1_weighted" in the model training. I didn't have time to re-run the code to unify the score metric, but the difference should be mild.

Code reference. Please refer to */text-document-classifier/04. Model Training* if you are interested in the tuning parameters for each method used in this project.

3 Numerical Results

Given a result for an estimate, it can be either true positive (TP), false positive (FP), true negative (TN) or false negative (FN). There are several metrics to measure the performance: accuracy, precision, recall, F1 score, area under the ROC curve and other. Among all, precision measures the "accuracy" of our selected set, defined as TP / (TP + FP) and recall measures the percentage that we pick the true ones, defined as TP / (TP + FN). In this project, we are required to use the F1 score, which is harmonic mean of precision and recall. We have F1 score for each category, defined as [2 precision * recall / (precision + recall)], and then we calculated the final F1 score weighted by the size of each category for comparison. I also added the accuracy as a reference, which is defined as (TP + TN) / (TP + TN + FP + FN). (I didn't store the training F1 score, so it is missing from the table.)

Model	F1 Score	Accuracy	
	Testing	Training	Testing
Multi-logit	63%	81%	63%
Naive Bayes	62%	74%	62%
Random Forest	59%	100%	59%
KNN	54%	100%	54%

From the table, the F1 score and accuracy are consistent and the multinomial logistic (multi-logit) achived the best testing F1 score and accuracy. Random forest and KNN were overfitting the data, and thus the testing results were not desirable. Naive Bayes had comparable performance with multi-logit, and it was taking much less time. We can take a closer look at the confusion matrix of the multinomial logistic model using testing data (Figure 4), where the large values in the diagonal show its good performance.

4 Conclusion

In this project, we constructed a text classifier from the original documents using a simple approach, and achieved prediction accuracy and F1 score up to 63%. Due to the time limit, there are still possible improvements. One of those is to represent data using more strategies and thus create more features. Another one is that one can tune hyper parameters using a more finely-defined grid to generate better hyper parameter if having time budget. Last but not the least, there are other potential useful information to extract from the original document, such as email address, that I didn't take the full advantages of. Overall, it is an exciting project and the result definitely has room to improve.

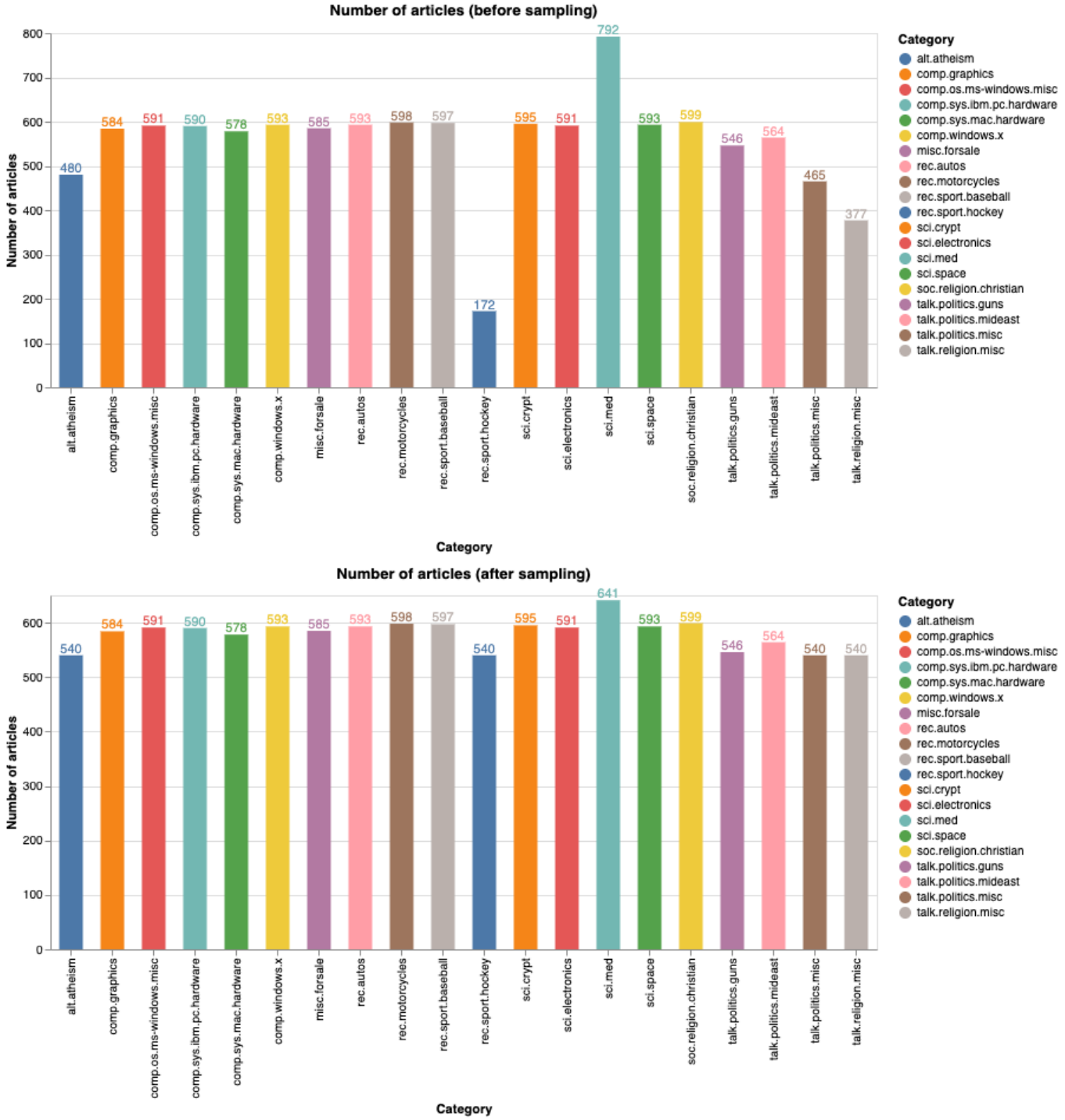

Figure 2: Comparison the sample size per class before and after sampling in the training data.

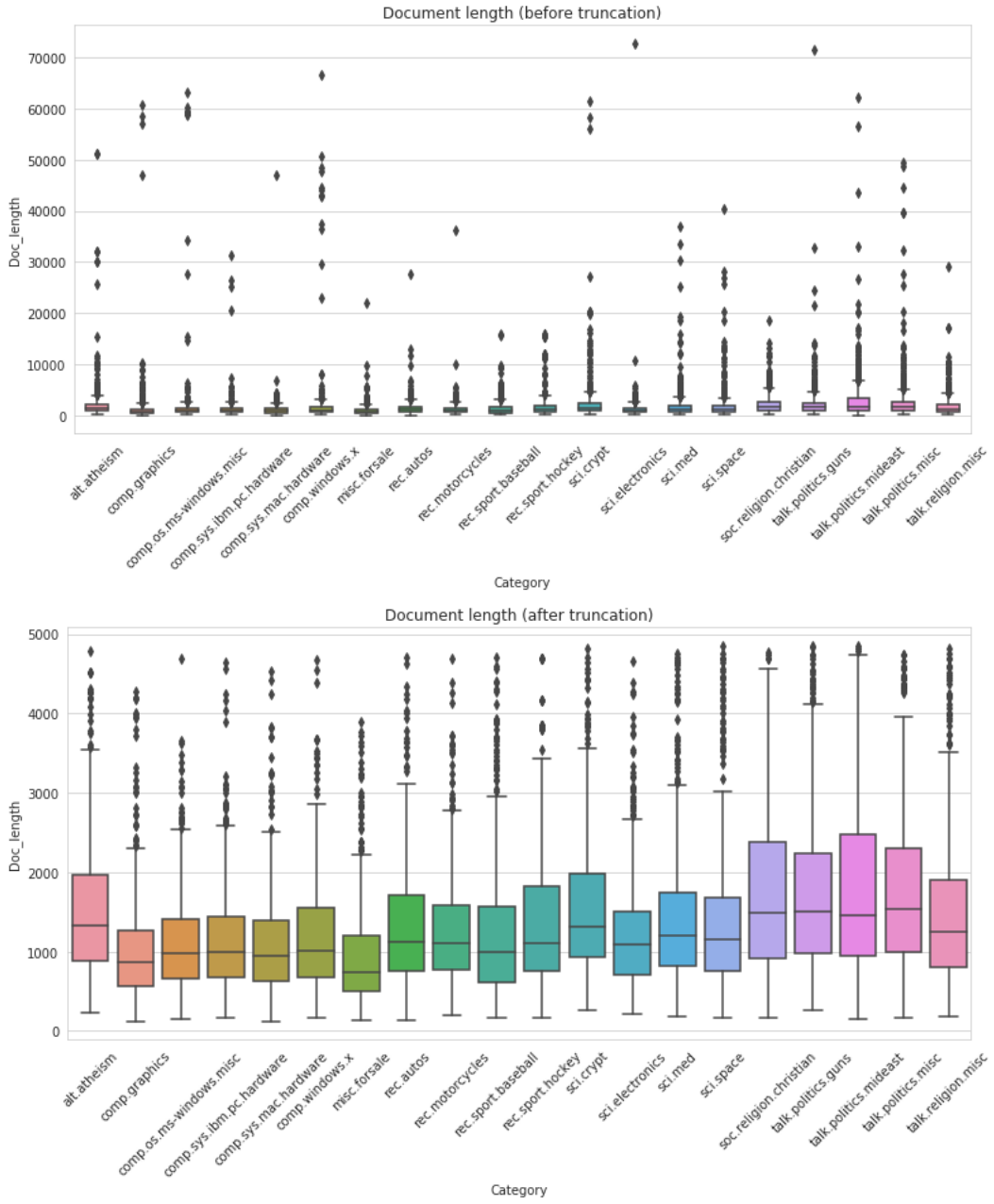

Figure 3: Comparison the document length per class before and after truncation in the training data.

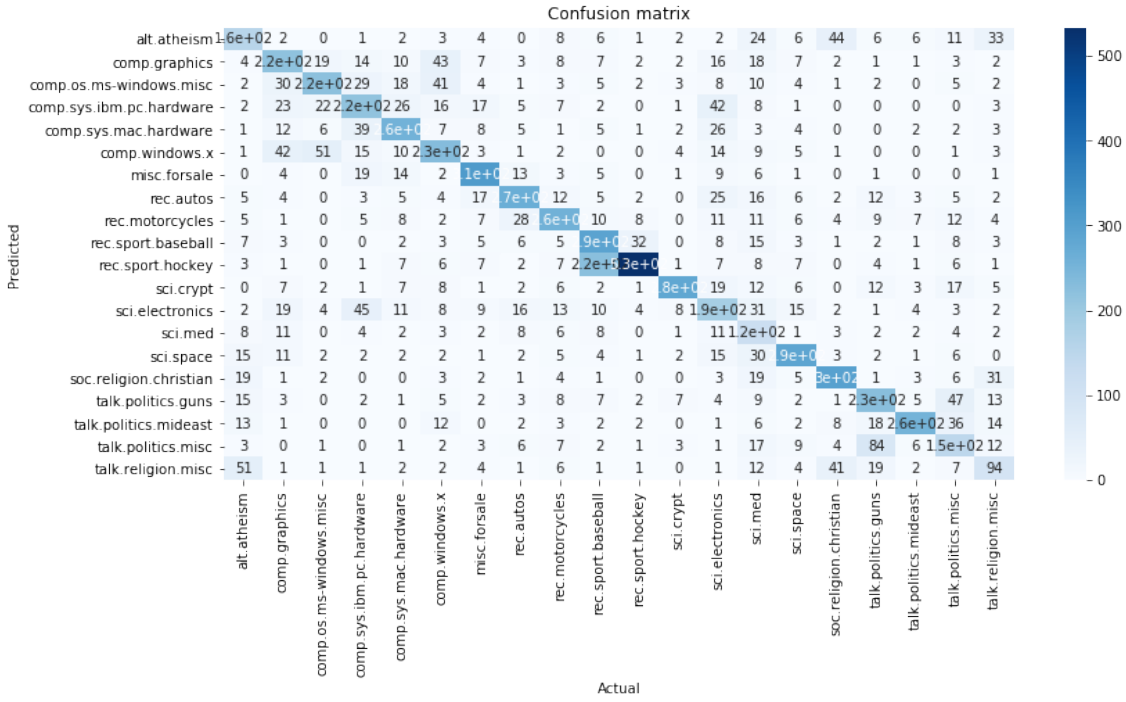

Figure 4: Confusion matrix of the multinomial logistic model (test data).