Stance Detection and Stance Classification

Group Report for Capstone Spring 2019
Data Science @ George Washington University
Xiaodan Chen, Xiaochi Li
Code: https://github.com/FiscalNote/FN-Research-GW-Opinion-Mining

Abstract

Nowadays, more and more companies are using the insights of political information to make right decisions. However, most documents are unstructured or in human language, which costs the companies tremendous resources to get something useful.
This project proposed a two-stage model to do stance detection (by Xiaodan Chen) and stance classification (by Xiaochi Li) sequentially. We examined the quality of the data and handled the mislabeled issue and imbalanced issue for stance detection and stance classification respectively. The project evaluated the performance of Deep Neural Network and conventional machine learning model in stance detection problem, and tested the effect of several different vectorization, preprocessing and oversampling on the performance of Random Forest, XGBoost, Support Vector Machine and Logistic Regression. The data is provided by FiscalNote, an information technology company located in Washington D.C., and the project will be integrated into the company's production system.
The best model for stance detection is pretrained GloVe + LSTM with F1 score of (0.86, 0.96), the best model for stance classification is Tfidf + Lemmatization + SMOTE + Logistic Regression with F1 score of (0.75, 0.93).

Keywords: Natural Language Processing, Machine Learning, Computational Political Science, Stance Detection, Stance Classification

1. Introduction

We are in the era where very large amount of politics related documents can be easily accessed on line. The availability of such data is made by the trend of "every-body putting their records on the Internet". It not only includes the full text of laws and regulations, but also the official process of legislation, such as the speeches in the congress or unofficial documents like the politician's opinion in news, blogs or tweets. While more and more companies are trying to take advantage of these publicly available politics data to get insight of legislation process before the law is made, the first obstacle they face is that the data is too much to analyze manually.
Take the U.S. congressional record as an example, in the 113th congress, there are over 118 thousand speeches made in the congress. Only a small part of the speeches has stance (support or opposition) in them. Obviously, it's impossible to read all of them and write summaries about these speeches manually.

This paper focus on the fundamental problem of analyzing congressional speech, determining whether the speech contains a stance and whether the stance is support or opposition. We will utilize machine learning and natural language process techniques to build a two-stage model to solve stance detection and stance classification problem. And we will compare and evaluate the performance of various methods and find out the best one for implementation.

2. Related Work

Previous stance detection work includes that by Walker et al. (2012), Hasan and Ng (2013) and they focus mainly on debates; Lately, there is also a great amount of researches in Twitter stance detection(Mohammad et al., 2016; Liu et al., 2016; Sobhani et al., 2016); Recently, fake news has become a problem and has been studied a lot with regard to stance detection by credible media(Ruder et al., 2018; Hanselowski et al., 2018). Besides, the problem of imbalance data was considered. An ensemble method to resample data for both the majority and minority class for each tree in random forest was completed by Chen et al. (2004). For neural networks, Ye Zhang and Byron Wallace performed a sensitivity analysis into the hyperparameters needed to configure a single layer convolutional neural network for document classification. Zhang, et al. use a character-based representation of text as input for a convolutional neural network.

3 Proposed Framework

3.1 Data Source

The data is provided by FiscalNote, and it came from the congressional record of the 113th congress in 2013-2014. The data contained three parts, one is action_data.csv which is a csv file of stance taking actions by legislators, another is document_data.csv which is a file containing information about the congressional record data, and the last part is the cr_corpus folder which contained the congressional record stored in XML format, and structured as follows "/cr_corpus/{volume_no}/{issue_no}/{page_range}/{document_id}.xml". We used the shared columns in action.csv and document.csv (bold in Table 3.1, Table 3.2) to create a stance label- file location pair.

Column Name (type)	Description
action_text_string(str)	the stance of the legislator
cr_pages (str)	the pages of the congressional record that contain the stance
cr_date(datetime)	the date of the congressional record entry containing the stance
person_id (int)	internal id for the legislator taking the stance

Table 3.1 Part of the columns in action_data.csv

Column Name (type)	Description
document_id (int)	an internal document id
volume_no (int)	the volume of the congressional record
issue_no (str)	the issue number of the congressional record (subset of volume_no)
page_range (str)	the pages of the record containing the document (subset of issue_no)
pub_date (datetime)	date the congressional record was published

Table 3.2 Part of the columns in document_data.csv

After we located the right path to search, we used Beautiful Soup as the XML parser and filter out the speeches with person_id and stored it with the stance label as labeled data.
We also load all the speech that did not appear in action_data.csv as unlabeled data for further use. Finally, we got 118157 speeches from the 113th congress and 2077 of them have labels.
The data loading method is implemented in 'corpus_loader.py' and the xml parsing method is implemented in 'xml_parser.py'.

3.2 Method

The general method we use is two-stage model. We built a stance detection model to filter out the speech that may contain a stance, followed by a stance classification model to classify whether the stance is support or opposition. We developed both model in an iteration of "preprocess-modeling-evaluation", until we found the best model and moved on to deployment.

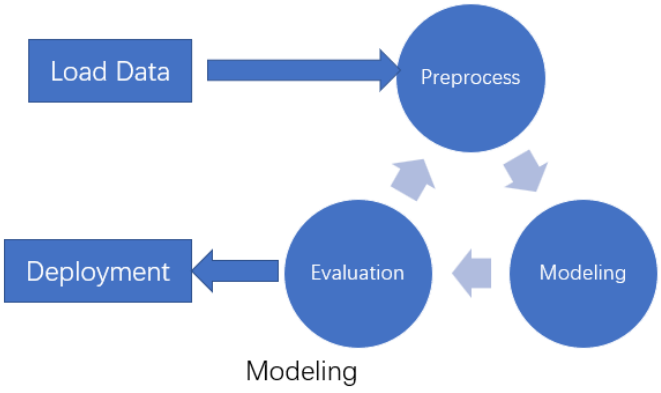

Figure 3.1 Process of model development

3.3 Proposed method for Stance Detection

3.3.1 Experiment Design

The method of our experiment is control variable, and the metric is F1 score for both classes in train set and test set. We will test all the possible combinations of three preprocessing options, vectorization, three balancing option and four models listed in Table 3.3 for stance detection.

Preprocessing	Vectorization	Balancing Data	Model
1. Remove non-alphabetic words	1. Tf-idf	1. Resampling	1. Logistic Regression
2. Remove stop words		2. Ensemble Method	2. Random Forest
3.Remove special characters		3. Relabeling	3. GloVe+LSTM
3.limit word frequency			4. GloVe+CNN+lstm

Table 3.3 Components of experiment for stance detection

3.3.2 Text Preprocessing and Vectorization

Text preprocessing, such as removing stop words, non-alphabetic words, special characters, limit word frequency, are common preprocessing methods in natural language process and applied in this paper. Stop words are a set of commonly used words in English, and they usually are non-informative, so they are often removed in preprocess. These methods were expected reduce memory overhead, reduce noise and potentially improve the performance of the models.

In addition, the machine learning models can't take words or sentences as input, so we must transform the speech into a feature vector to feed into the model. Tf-idf vectorization was applied for this project. Tf–Idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The Tf-Idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. Tf–idf is one of the most popular term-weighting schemes today.

3.3.3 Resampling and ensemble method for imbalance data

We firstly check our label data, as shown in Figure 3.2, the stance data is extremely imbalanced. Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. The reason behind it for this project is straightforward. In real world, human labeling for text is expensive, it's time consuming and cost consuming. However, new speeches and bills are generated every day. Thus imbalance data is common in the field of NLP. In this project, this problem could affect the result of machine learning model a lot since models tend to predict data to be always in the same class or group no matter what. There are a lot of ways in papers to alleviate the downside effect of imbalance data. In this paper, three methods would be tried respectively to avoid the problem, namely resetting class weights, resampling using ensemble method, and data check in depth again.

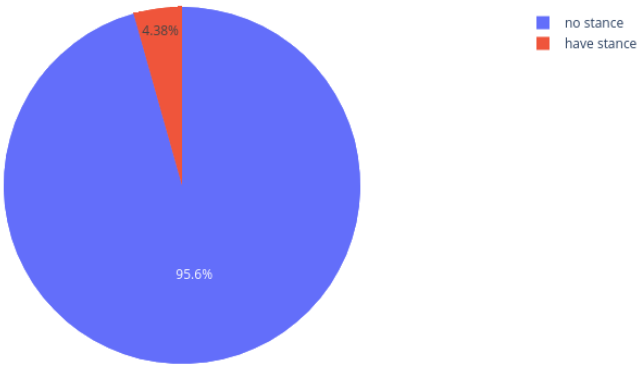

Figure 3.2 the distribution of the stance label data

For the first method of changing classes' weights in itself, sklearn has its own built-in method to recalculate the weights of different classes. But in default, all classes are supposed to have weight one. if you want it to be recalculated, you could set the parameter 'class_weight' to 'balanced', which uses the values of y to automatically adjust weights inversely proportional to class frequencies.

For the second method of resampling with ensemble method, it was actually inspired by one paper 'Using Random Forest to Learn Imbalanced Data' firstly. The author try to use ensemble methods to alleviate the downside effect caused by imbalance data and runs experiments that methods was shown to improve the prediction accuracy of the minority class for imbalance data issues, and have favorable performance compared to the existing algorithms. This method was already implemented by 'imblearn' and applied here for this specific NLP stance detection problem.

There are two ways that were tried, one is the balanced random forest ensemble method and the other is the weighted random forest method. The balanced random forest draws a bootstrap sample from the minority class and draw a bootstrap sample from the majority class with the same size of the minority sample. It also induces a classification tree to maximize size without pruning. the tree is induced with CART algorithm with the modification of only searching through a set of m randomly selected variables instead of searching through all variables to find the optimal split, finally aggregate the predictions of the ensemble and get the final prediction.

However, the weighted random forest places a penalty on misclassifying the minority class. they assign a weight to each class, for the minority class they give a larger weight (i.e. higher classification cost). The class weights are incorporated into the RF algorithm in two places. In the tree induction procedure, class weights are used to weight the Gini criterion for finding splits. In the terminal nodes of each tree, class weights are again taken into consideration. The class prediction of each terminal node is determined by "weighted majority vote"; i.e., the weighted vote of a class is the weight for that class times the number of cases for that class at the terminal node. The final class prediction for RF is then determined by aggregating the weighted vote from each individual tree, where the weights are average weights in the terminal nodes. Class weights are an essential tuning parameter to achieve desired performance. The out-of-bag estimate of the accuracy from RF can be used to select weights.

Unfortunately, for this specific project, the F1 score is still pretty low for the minority class for the two methods we tried before, which gives very little improvement for both the majority class 'no stance' and the minority class 'have stance', we realized that these two ways are not feasible here and decided to go back to check our data again.

3.3.4 Relabeling

Since it's obviously that resampling, and class weight method doesn't work like a charm here. Thus, it's a good idea to go back and check data manually. The finding is a surprise here. A lot of speeches in bills and transcripts with some strong indications of stances are mislabeled as '0', namely no stance. For this case, we decided to come up with regular expression relabeling model. By detecting some strong key words, such as 'support', 'oppose' and so on, the relabeling model would relabel all originally unlabeled data. The result is as following in Figure 3.3.

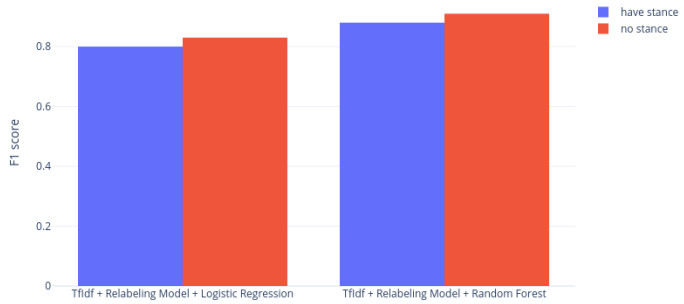

Figure 3.3 the F1 score performance after relabeling

So far, We've realized that the true problem doesn't lies on imbalance data because the imbalance data is not caused randomly but instead intentionally. When we've improved the data quality using the relabeling algorithm, the performance for both minority and majority class has improved a lot.

3.3.5. Models

As for machine learning part, we've only adopted logistic regression and random forest for this project.

As for deep learning part, there are two structures, one for LSTM with pretrained GloVe word embedding. This structure involves the use of a pretrained word embedding for representing words, a LSTM for learning how to dicriminate stances on classification problems. Yoav Goldberg, in his primer on deep learning for natural language processing, comments that neural networks in general offer better performance than traditional machine learning methods, for example, classical linear classifiers, especially when used with pre-trained word embeddings.

Another one neural network structure adds an additional convolutional layer compared with the first structure. This new Convolutional Neural Network (CNN) is to extract high level features compared to the structure above. It's more like a ngram method in traditional NLP problems. Here in this project, the kernel size of the convolutional layer is 3, which means it could extract 3 words together at a time to do prediction.

One thing needs to mention is that when I was working on the keras to build neural network, I tried to limit the word frequency in speeches with 5000 words of high frequency. I read the documentation of keras carefully and set one parameter 'num_words' in Tokenizer to be this number, but it didn't work. The vocabulary size is still way more large, then I checked the source code of keras and did some research online, then I realized it's the problem of keras itself. The developer hasn't fixed this bug, so I write a function to limit the word frequency in advance before using keras for this NLP project this time. I really learned that we can not always count on things what we think it should be always right, or we would waste a lot of time getting stuck in the same place. Figure 3.4 below show the two structures of neural networks, and Figure 3.5, Figure 3.6 below show the performance of these two structures of neural networks

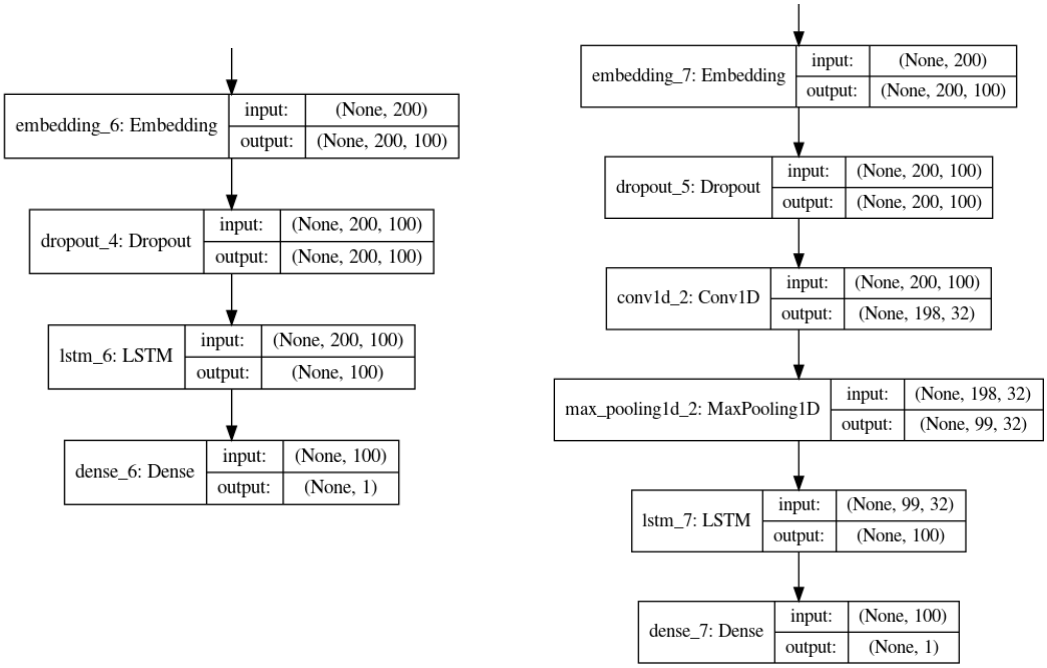

Figure 3.4 neural network structures

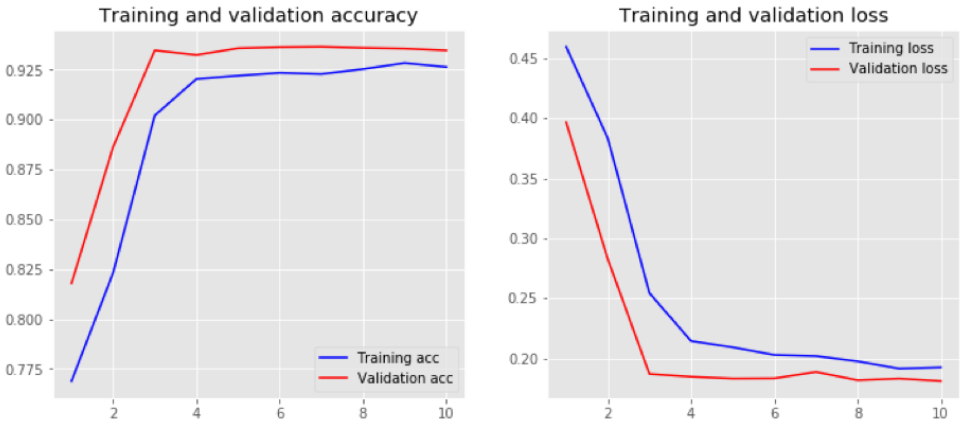

Figure 3.5 Accuracy and loss for structure 1(embedding + LSTM)

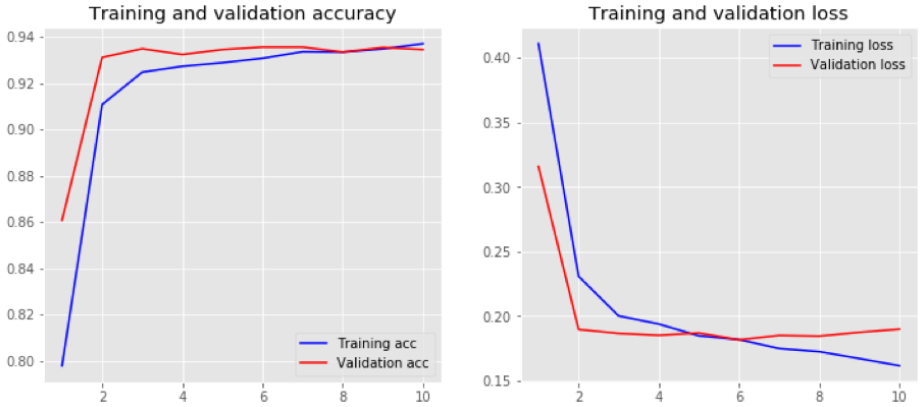

Figure 3.6 Accuracy and loss for structure 2(embedding + CNN + LSTM)

3.4 Proposed method for Stance Classification

3.4.1 Experiment Design

The method of our experiment is control variable, and the metric is F1 score for both classes in train set and test set. Beside the F1 score, we also care the speed to train and interpretability of the model. We will test all the possible combination of three preprocessing options, two vectorization options, two balancing option and four models listed in Table 3.4

Preprocessing	Vectorization	Balancing Data	Model
1. Do nothing	1. Count Vectorization	1. Do nothing	1. Logistic Regression
2. Remove stop words	2. Tf-idf Vectorization	2. SMOTE	2. Random Forest
3. (2) +Lemmatization			3. SVM
			4. XGBoost

Table 3.4 Components of experiment for stance classification

3.4.2 Preprocessing

As shown in Figure 3.7, we found that 75% of the speeches has less than 439 words, so we set a cutoff of 500 words to avoid over long speech's' negative impact on model's performance.

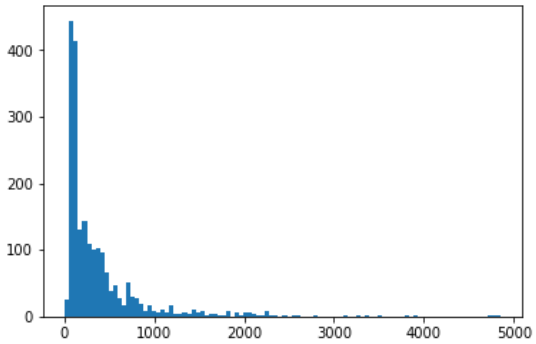

Figure 3.7 the length of the speech

Remove stop words and lemmatization are common preprocessing methods in natural language process. Stop words are a set of commonly used words in English, and they usually are non-informative, so they are often removed in preprocess. Lemmatization means to revert a word to its basic form. Both methods are expected reduce memory overhead, reduce noise and potentially improve the performance of the models. However, we can not take for granted that this expectation is correct, so we still compare these two methods with doing nothing in preprocess.

3.4.3 Vectorization

The machine learning models can not take words or sentences as input, so we must transform the speech into a feature vector to feed into the model. We used "Bag of Words" representation in the experiment, and there are two vectorization methods, count vectorization and Tf-idf vectorization.

Count vectorization will transform a speech into a vector $\vec{A}$ where element $A_i$represents the occurrence of i^{th} word in the feature list. Count vectorization is a convenient way to vectorize the speech, however it did not consider the frequency of words across speeches.

Tf-idf which stands for Term Frequency Inverse Document Frequency is an improvement to the count vectorization.

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document} \quad (3.1)$$

$$IDF(t) = log_e\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it}\right) \quad (3.2)$$

$$TF - idf\ score = TF \times IDF \quad (3.3)$$

As indicated in formula 3.1, 3.2, 3.3, the major difference is IDF, which is a score of how rare the word is across all speeches. Rarer the term, more is the IDF score.
We expect the tf-idf vectorization reduce noise of common words across support and opposition speeches, because such words are expected to have a low IDF score.

3.4.4 Balancing the data

As shown in figure 3.8 shows, less than 1/4 of the labeled samples are negative. And we can expect the imbalanced data will cause a low F1 score on the minor class. In order to examine whether balancing the data would help, we used SMOTE as a over-sampling method to compare with taking no action on the train set.

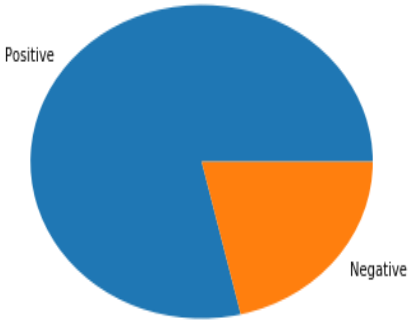

Figure 3.8 Label Distribution

SMOTE stands for Synthetic Minority Over-sampling Technique, it's an unsupervised method that synthetic new samples for minority class based on existing nearby samples. As shown in figure 3.9, new samples are synthetized(red points) between exiting samples.

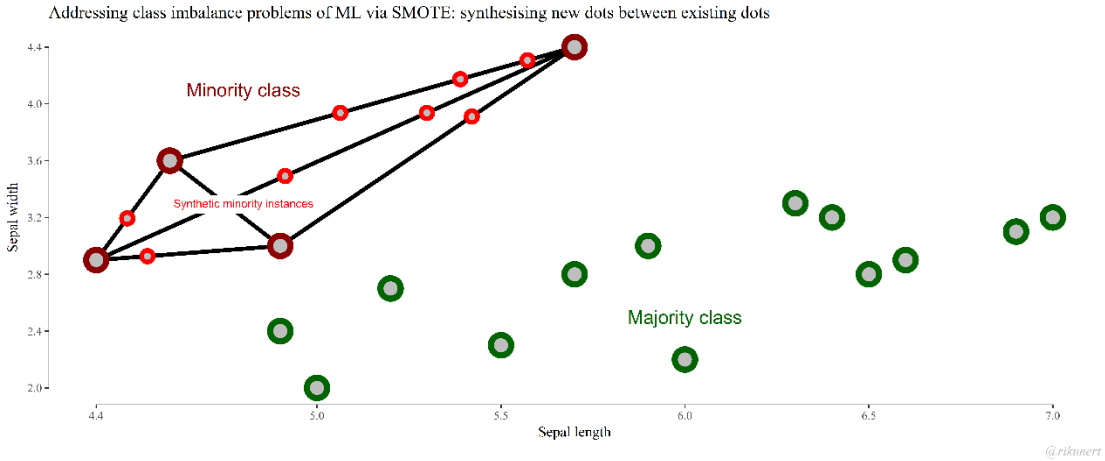

Figure 3.9 How SMOTE synthetic new samples[1]

3.4.5 Models

We will use Random Forest, XGBoost, SVM and Logistic Regression in the experiment.
Random Forest and XGBoost(eXtreme Gradient Boosting) are ensemble learning methods based on decision trees. We already know about Logistic Regression and SVM, so we will discuss about the new method, XGBoost.
The difference is that the random forest train trees separately while XGBoost train trees consecutively. And we can expect random forest has a lower variance while XGBoost has a lower bias.

[1] SMOTE explained for noobs - Synthetic Minority Over-sampling TEchnique line by line (http://rikunert.com/SMOTE_explained)

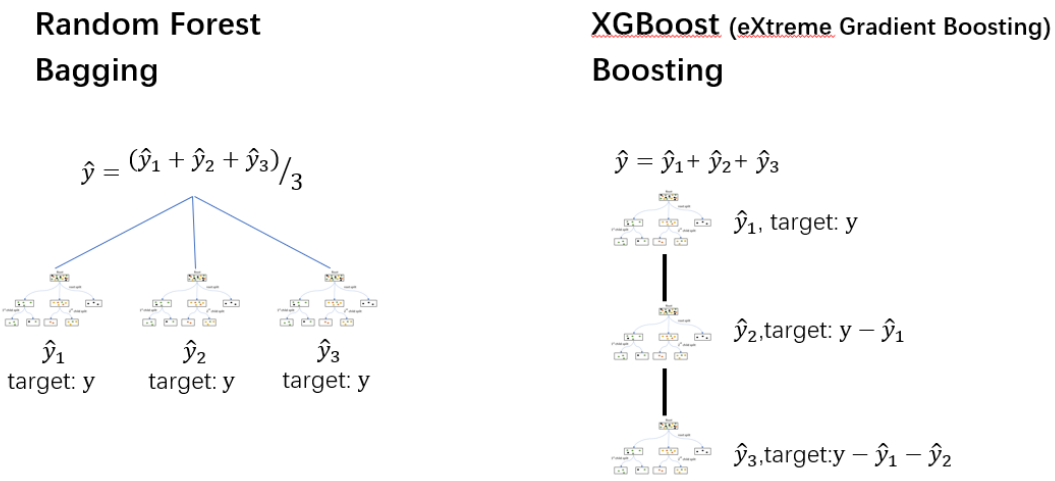

Figure 3.10 Comparison of Random Forest and XGBoost

The Support Vector Machine in the experiment uses linear kernel for convenience, and we did not tune logistic regression model.

4. Experiment Result

4.1 Stance Detection

4.1.1 Finding on Preprocessing, and Balancing data

We found that balancing the data with relabeling is the most effective method to improve model performance since the imbalance data is caused intentionally. Remove non-alphabetic words contributes a lot as well and sometimes more text preprocessing doesn't equal a better result.

4.1.2 Model comparison

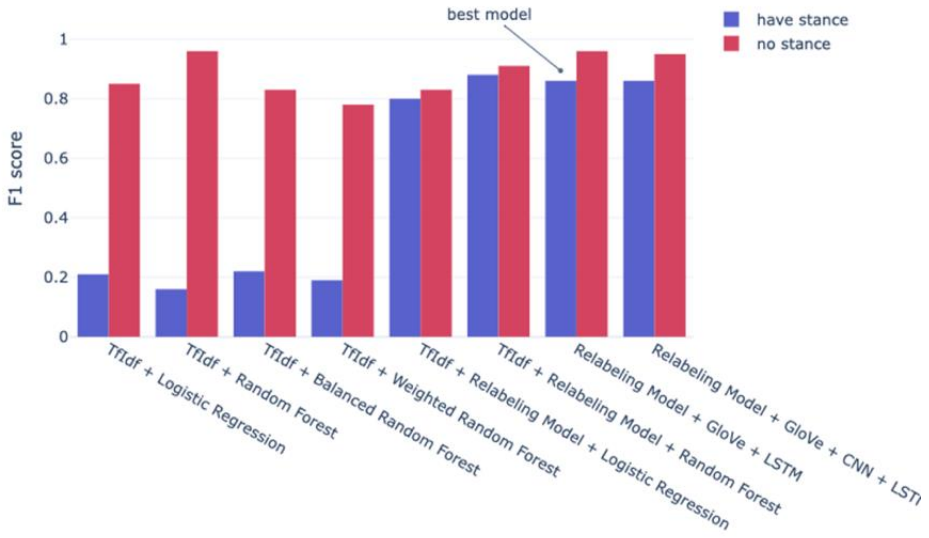

Figure 4.1 Model Comparison

Combination/Metrics	F1 score (have stance)	F1 score (no stance)
TfIdf + Logistic Regression	0.21	0.85
TfIdf + Random Forest	0.16	0.96
TfIdf + Balanced Random Forest	0.22	0.83
TfIdf + Weighted Random Forest	0.19	0.78
TfIdf + relabeling model + Logistic Regression	0.80	0.83
TfIdf + relabeling model + Random Forest	0.88	0.91
Pretrained GloVe + relabeling + LSTM	0.86	0.96
Pretrained GloVe + relabeling + CNN + LSTM	0.86	0.95

Table 4.1 Summary of F1 score of all models for stance detection

Model	Performance	Interpretability	Speed
TfIdf + Logistic Regression	✗	✓	✓
TfIdf + Random Forest	✗	✓	✓
TfIdf + Balanced Random Forest	✗	✓	✓
TfIdf + Weighted Random Forest	✗	✓	✓
TfIdf + relabeling + Logistic Regression	✓	✓	✓
TfIdf + relabeling + Random Forest	✓	✓	✓
GloVe + relabeling + LSTM	✓	✗	✗
GloVe + relabeling + CNN + LSTM	✓	✗	✗

Table 4.2 Summary of models in experiments

After taking comprehensive considerations of a combination of performance, interpretability and speed, we think the random forest model with the relabeling algorithm is the best model here. However, if we only consider performance here, the LSTM with pretrained GloVe embedding would be the best one.

4.2 Stance Classification

4.2.1 Finding on Preprocessing, Vectorization and Balancing data

We found that balancing the data with SMOTE is the most effective method to improve model performance, and it works well when we use Tf-idf vectorizer. Remove stop words contributes more than lemmatization. Lemmatization is the slowest part in preprocessing, and it's also the least effective part, so we would not set it as a default procedure in implementation.

4.2.2 Model comparison

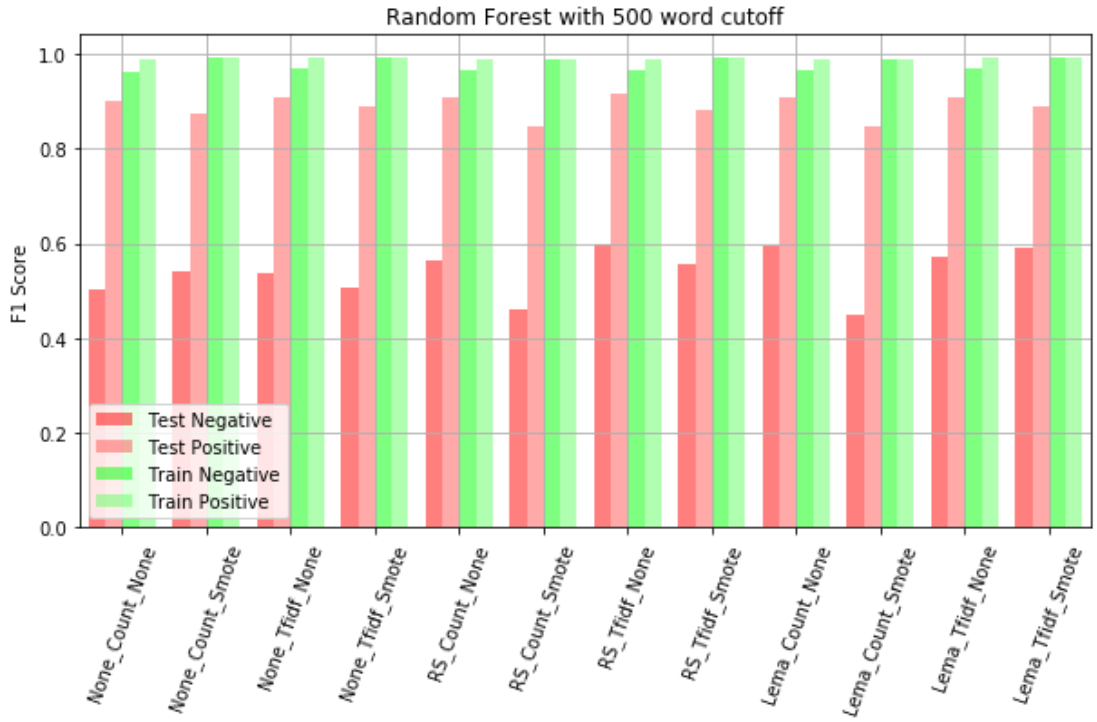

Figure 4.2 Experiment result of Random Forest

As shown in Figure 4.2, the random forest did not perform well on the test set, and it is still influenced by the imbalanced problem. The F1 score on train set is almost 1, which suggests that the model is over-complicated and is causing over-fitting.

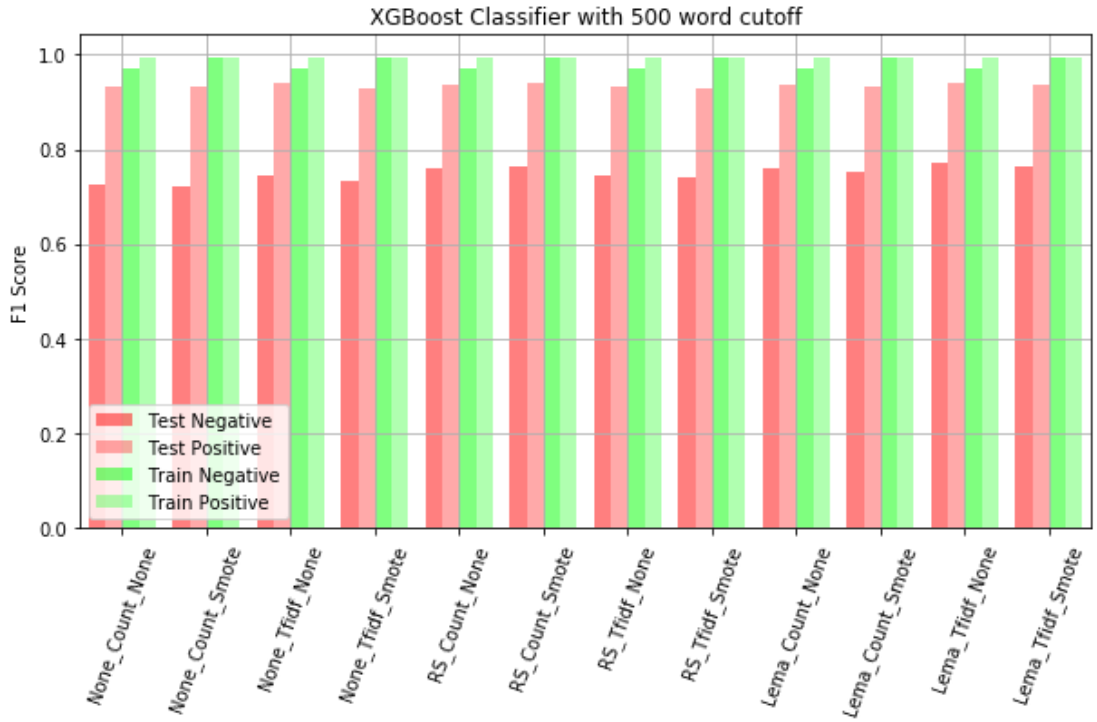

Figure 4.3 Experiment Result of XGBoost

As shown in Figure 4.3, the XGBoost performed better than Random Forest, and it suggests that boosting is a better strategy in this situation.

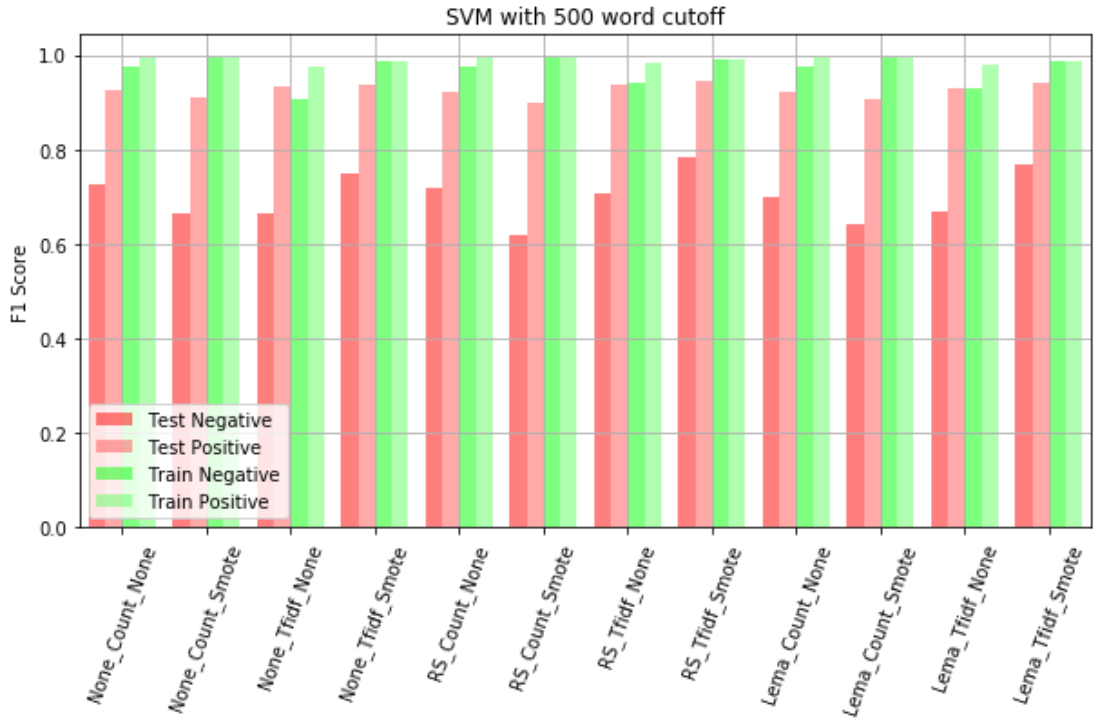

Figure 4.4 Experiment Result of SVM

As shown in Figure 4.4, the SVM model performance relatively better when the data is imbalanced. However, it takes 11 times more time to train a SVM model comparing with other models like Logistic Regression. Thus, it's not applicable to train on a much larger data set in the future in real implementation.

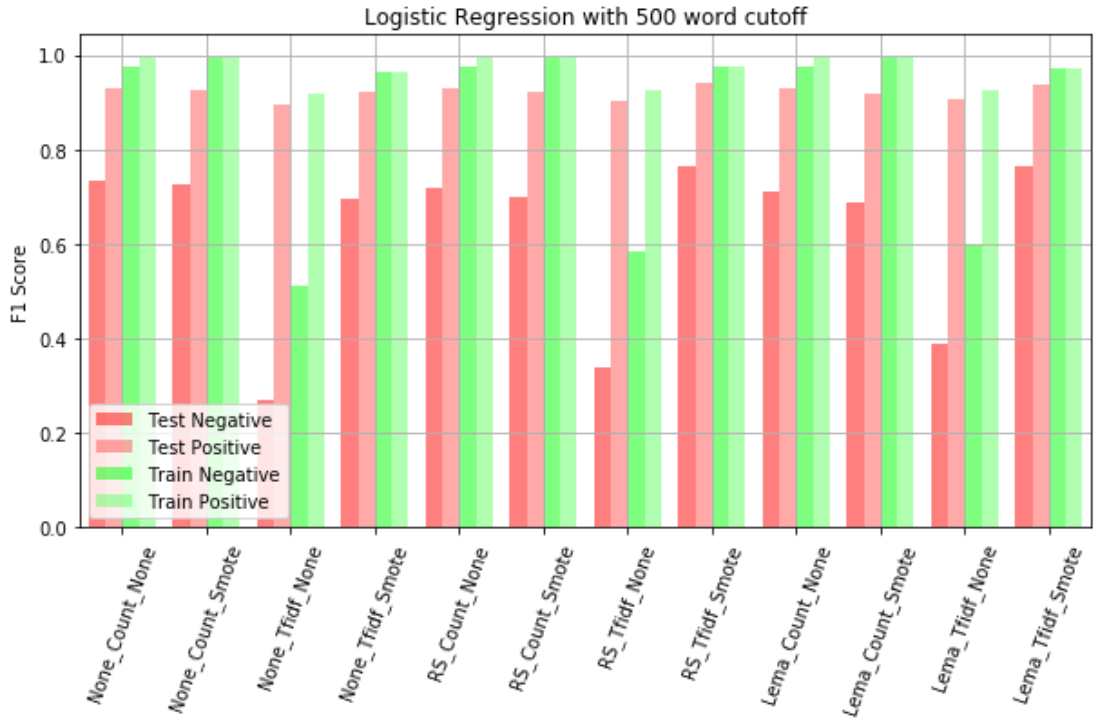

Figure 4.5 Experiment Result for Logistic Regression

As shown in Figure 4.5, Logistic Regression has a similar performance with XGBoost and SVM when the train set is balanced.

Model	Test Negative	Test Positive	Train Negative	Train Positive
Random Forest	0.588	0.894	0.992	0.992
XGBoost	0.765	0.935	0.992	0.992
SVM	0.768	0.941	0.987	0.987
Logistic Regression	0.756	0.936	0.969	0.968

Table 4.3 Summary of F1 score with Lemmatization and SMOTE as controlled variable

Model	Performance	Interpretability	Speed
Random Forest	✗	✗	✓
XGBoost	✓	✗	✓
SVM	✓	✓	✗
Logistic Regression	✓	✓	✓

Table 4.4 Summary of models in experiments

After taking performance, interpretability and speed into consideration, we found that Logistic regression has better interpretability than XGBoost and takes fewer time to train comparing with SVM. So, we selected Logistic Regression as our model for stance classification task.

5. Deployment

After we discussed with our mentor from FiscalNote, we realized that the stance detection and stance classification model we built in this project will be integrated into a system that also has function of person/bill detection. The system, as shown in figure 5.1, will take the XML as input and generate person-stance-bill pair for further analysis such as policy analyze, policy alert and stance clustering.
There are two required functions for deployment, one is predicting a speech with stance detection and stance classification model, and another is retraining the model when more data is available.

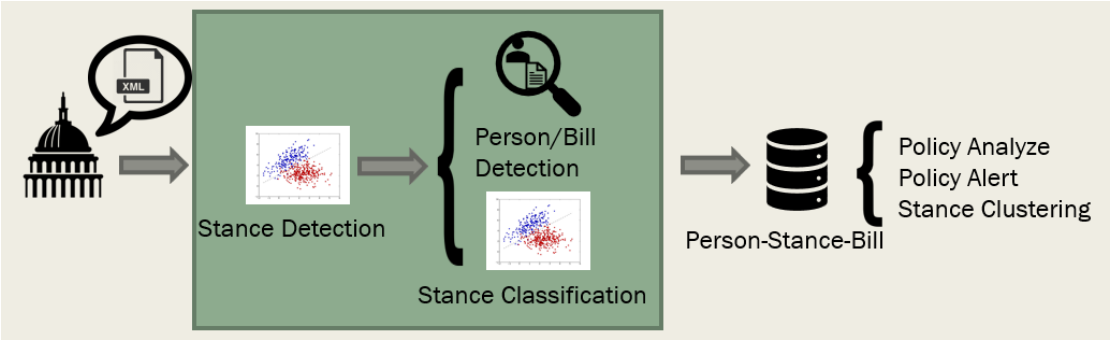

Figure 5.1 Schematic diagram of how the system works

6. Learning and conclusion

6.1 Best Model

The best model we have for stance detection is the pretrained Glove embedding+ LSTM
The best model we have for stance classification is Tfidf + Lemmatization +SMOTE + Logistic Regression.

6.2 Learnings

The main challenge we have is that the data quality is poor. The labeled data is unbalanced, and the unlabeled data cannot be used to train stance detection model.
We tried to tune the models, but we got only a slight improvement. Then we focused on improve the data quality and got some improvement. We learnt that the quality of data is more important than the complexity of the model. And we cannot omit the process of checking the data before using it.

Acknowledgements

At the end of this paper, we would like to thank Professor Brian Wright and Dr. Vladimir A. Eidelman for offering us this great opportunity to complete the capstone in FiscalNote.
Also, we are thankful to Dr. Daniel Argyle as our mentor during the project.
We would also thank all the professors in the Data Science program at the George Washington University, without whom we would never learnt so much in two years.