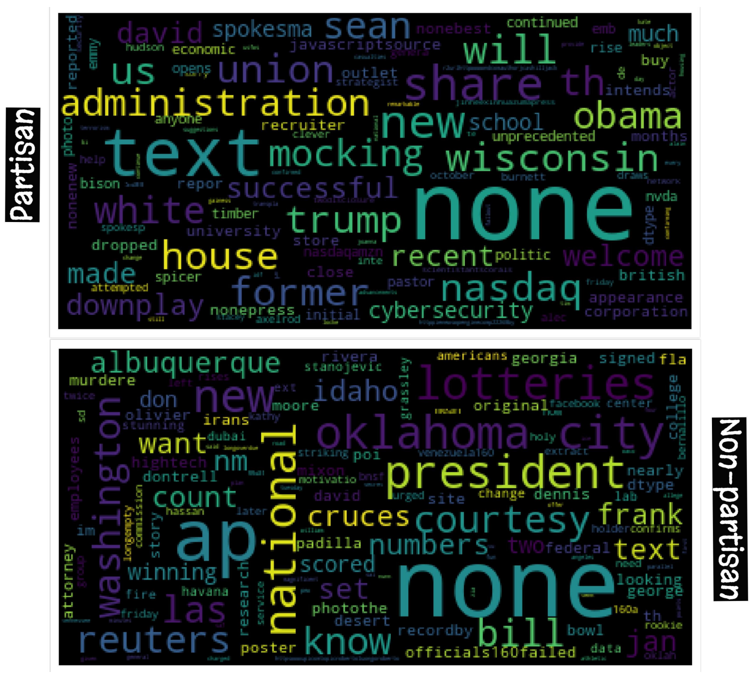
HYPERPARTISAN NEWS DETECTION

Project Report

Team: District 13

Yash Deshpande	yd1282@nyu.edu
Yada Pruksachatkun	yp913@nyu.edu
Aja Klevs	ak7288@nyu.edu
Jiayi Du	jd4138@nyu.edu

December 7, 2018



Contents

CHAPTER 1 | BUSINESS UNDERSTANDING

During the 19th century, newspapers had trouble making a profit through sales alone. Instead, party sponsorship and political donations were a main source of income. As a result, reporters often used politicians and party officials as their primary sources, leading to the pervasion of highly partisan news outlets. In turn, this often catalyzed excessive political animosity and allowed political figures to distort the truth in order to manipulate the public.

The advent of muckraking in the early 1900s dramatically increased the market for news consumption. Because of this, news outlets increasingly transitioned to subscriptions, sales, and advertisements as their primary source of income. In this manner, muckraking eventually introduced the concept of a neutral press. The dissociation of mainstream newspapers and political parties has had wide-sweeping consequences in the past century. Coverage of FDR's court-packing plan, the Watergate break-ins, and the George W Bush administration's torture policies are just a few examples of important journalism made credible by the non-partisan nature of the mainstream press.

However, it isn't all smooth sailing. While a neutral press has grown into its market, partisan news sources on both the left and right still persist. Some examples are Breitbart news and Infowars on the right, or the Nation and Democracy Now! on the left. Such news sources often spread propaganda which reinforce biases, and can lead to an ill-informed and divided public.

In such an environment, it becomes increasingly necessary for an effective way to determine whether or not a news partisan argument, which is exactly what this project aims to do. This is in the interest of both publishing houses, who would want to maintain their nonpartisan image, and consumers, who would be able to keep themselves better informed about politics and news.

To work toward an effective solution to this problem, we tried multiple approaches- *logistic regression*, *SVM*, and *Naive Bayes*, all of which have demonstrated reliability and robustness in text classification problems. However, we found that due to the differences in the distributions of the training and test datasets, neither of these approaches were able to generalize well. They had a tendency to significantly overfit the training data. Due to this, we tried a neural network approach, which, instead of merely considering term frequencies and the distribution of the training data, learns and encodes for context and/or meaning.

CHAPTER 2 | DATA UNDERSTANDING

The data used to train the various models deployed herewith were obtained from an annual competition titled SemEval. The data and various features that were extracted from it are described below.

The data were made available to us in *.xml* format and as two separate files, as listed below.

- **Articles**

  Training: A set of $600,000$ news articles, prefixed with ID tags.

  Testing: A separate set of $150,000$ news articles, also prefixed with ID tags.

- **Labels**

  A set of ID tags and their respective labels. The label, titled '*Hyperpartisan*', is a binary label ($True/False$) indicating whether or not the article corresponding to the given ID follows a hyperpartisan argument. Each article in both datasets has been labelled based on the overall bias of its publisher, as reported by *BuzzFeed* journalists or *MediaBiasFactCheck.com.* Additionally, no publisher that appears in the training set appears in the test set, to ensure a better estimation of out-of-sample model performance.

The two datasets above were merged on the basis of the ID tags, and a full dataset containing *article ID, article text,* and *Hyperpartisan label* was created. For the purposes of this project, the full dataset was then limited to $20,000$ articles as a training/validation set, and $2,000$ articles as a testing set.

The full dataset of $600,000$ articles was balanced, i.e. half of the articles were labeled as following a hyperpartisan argument, and the other half as not. To ensure selection bias does not interfere with the results of our models, we ensured that our subset of $20,000$ articles was also balanced. Specifically, 50.33% of the articles in our training dataset follow a *hyperpartisan* (*i.e.*

biased) argument, and 49.67% do not. In the testing dataset, these percentages are 49.3% and 50.7% respectively.

CHAPTER 3 | DATA PREPARATION

3.1 Data loading

Since the dataset was relatively large and in an *.xml* format, the *xmltodict* Python package was used to parse both the articles and labels data. This package facilitates the parsing of large volumes of *.xml* data into dictionaries, with *xml* tags retained. During parsing, *IDs* and *articles* were retained using their tags, and any additional data such as *article links* and other *xml* tags were removed. The resulting dataset was then written to a *pandas* dataframe. Subsequently, a sample of $20,000$ was extracted, checked for selection bias, and unloaded to a *.csv* file to be used as our training dataset.

The same procedure was also followed for the test data, after which a sample of $2,000$ was extracted, checked, and unloaded to be used as our testing dataset.

3.2 Pre-processing

Once the training and testing datasets were loaded and transformed into the required format, pre-processing was required. This consisted of the following:

- **Removing stop-words**: English stop-words such as *the*, *is*, *at*, *which*, *and*, etc. were removed.
- **Removing punctuation**: All punctuation was removed.
- **Removing tags and URLs**: This was done to ensure tags that the *xml* parsing might have missed were removed. Long strings of unpunctuated URLs were also removed.
- **Case correction**: All words were changed to lower case.

3.3 Feature extraction

The textual data was converted to various features before modeling. These features have been described below.

- **Word and n-gram frequencies**: The text was vectorized using both *count* and TF/IDF (term frequency/inverse document frequency) vectorization methods. *n-gram* ranges from $(1, 2)$ to $(1, 6)$, along with $(6, 10)$ were tested during model selection. These features were incorporated into the basic models, namely, *logistic regression* and *Naive Bayes* approaches. In the subsequent chapter, we discuss the reasoning behind our final choice for the *n-gram range* in these models, which was the range $(1, 6)$.

- **Word embeddings**: For the neural network approach, *word embeddings* were used instead of word/term frequencies. Word embeddings are vectorized representations of words, which are learned as part of the model. In order for these representations to be useful, some empirical conditions should be satisfied; for example, vectors representing words that are similar in meaning should be closer to each other in the vector space. During the training process, these reprsentations are constantly changing in order to optimize the performance of the network. The maximum vocabulary size was set to be $20,000$. The first $20,000$ uni-grams are selected according to the gram frequency. *Tokens* (the aforementioned uni-grams) in the dataset are converted to *IDs* using a dictionary that maps each token to an ID value. Since neural networks prefer well-defined and fixed-length inputs and outputs, articles are padded to normalize length. To facilitate this, the mean value for the length of each article in the dataset is calculated. Additionally, the full training dataset is transformed into mini batches, so that articles can be fed to the network parallelly. This is done in order to optimize computation time.

The target variable is a binary label, titled '**Hyperpartisan**'. This variable, presented in the form of a Boolean ($True/False$), was converted to binary (0/1). '**1**' labels correspond to articles that follow a hyperpartisan argument.

CHAPTER 4 | MODEL SELECTION AND EVALUATION

Once the data was pre-processed and ready to work with, we decided to build, test, and evaluate three types of baseline models, namely, *logistic regression*, a *Support Vector Machine (SVM)*, and *Naive Bayes* classification. All of these models, as previously mentioned, used one or both of *count* and TF/IDF vectorization methods. For our final model, we built and trained a neural network, making use of *word embeddings* instead of *word/n-gram frequencies.* All the models detailed below were built on a subset of 70% $(14,000$ articles) of the full training set, and validated on the remaining 30% $(6,000$ articles). This split was facilitated using *scikit-learn*'s *train_test_split* function.

4.1 Logistic regression (LR) and Support Vector Machine (SVM)

To optimize our work based on the time constraint, for these two models, we tuned paramaters on a smaller subset of 5000 articles. An *n-gram range* $(1,4)$ was used during this process. However, once the full set of $20,000$ was used, we decided to extend this range to $(1,6)$, given that the Naive Bayes models (discussed subsequently) seemed to perform better for this range. *Count* and *TF-IDF* vectorizers were used for both *LR* and *SVM* approaches, along with a range of 7 different *L2 regularization weights.* The weights tested range from 10^{-30} to 10^{+30}. The results (AUC values and accuracies) from this parameter tuning can be found in Appendix 1- please see table 6.1.

Our parameter tuning revealed that the SVM models were more expensive in terms of time, and that varying the regularization weights did not affect their AUC values. Across all weights tested, the *count* vectorized SVM had an AUC value of 0.864 and the *TF-IDF* vectorization of 0.916.

In contrast, the performance of the *LR* models was highly dependent on the regulartization weights, with higher weights resulting in higher AUC values. The best performance, however, was obtained with *count*-vectorization coupled with a regulartization weight of 1. For the *TF-IDF* vectorized model, the regularization weight corresponding to the highest AUC value was 10^5.

Since *LR* with count vectorization resulted in the best performance, we decided to fine-tune the aforementioned parameters to ensure that the optimal regularization weight had been found. To do this, count-vectorized LR models with 19 different configurations were trained, using regularization weights ranging from 10^{15} to 10^{33}. The plot below shows the variation in validation set AUC values and accuracies with the regularization weights.

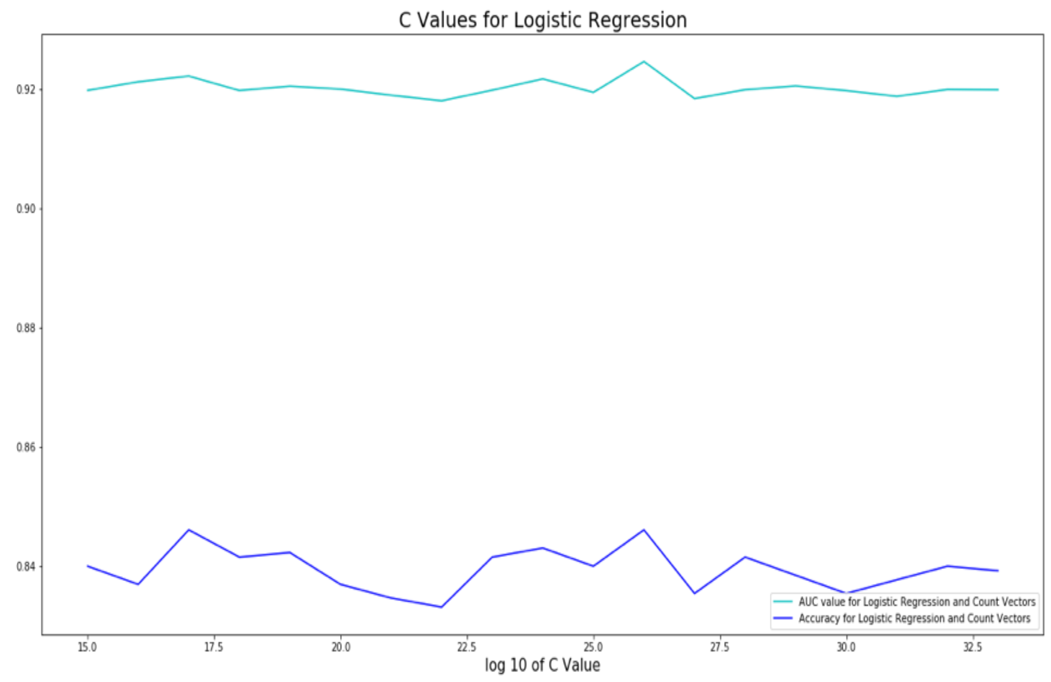


Figure 4.1: Variation in validation AUC and accuracy with regularization weights

The optimal regularization weight was hence determined to be 10^{26}. In conclusion, the best performing models from among the various configurations tested different categories, were, in increasing order of performance:

- SVM with *TF-IDF* vectorization and regularization weight = 1
- LR with *TF-IDF* vectorization and regularization weight = 10^5
- LR with *count* vectorization and regularization weight = 1

- LR with *count* vectorization and regularization weight $= 10^{26}$

The models listed above were then trained on the larger training set $(14,000)$ articles, and vectorized with an *n-gram* range of $(1, 6)$. The performance of these models on the validation set can be found in Appendix 1- please see figure 6.1. The LR model with a regularization weight of 1 performed the best, with an AUC of 0.937 and an accuracy of 86.8%. The performance on the test set, however, was not comparable, and a maximum test accuracy of 50.2% was achieved. ROC curves for the test data are slwo shown below.

Model	**Regularization weight**	**Test AUC**	**Test accuracy**
LR (count vec.)	1	0.501	50.00%
LR (count vec.	10^{26}	0.501	49.60%
LR (TF-IDF vec.)	10^{5}	0.502	50.20%
SVM (TF-IDF vec.)	1	0.501	50.10%

Table 4.1: LR/SVM- Model Performance on test set for n-gram range (1, 6)

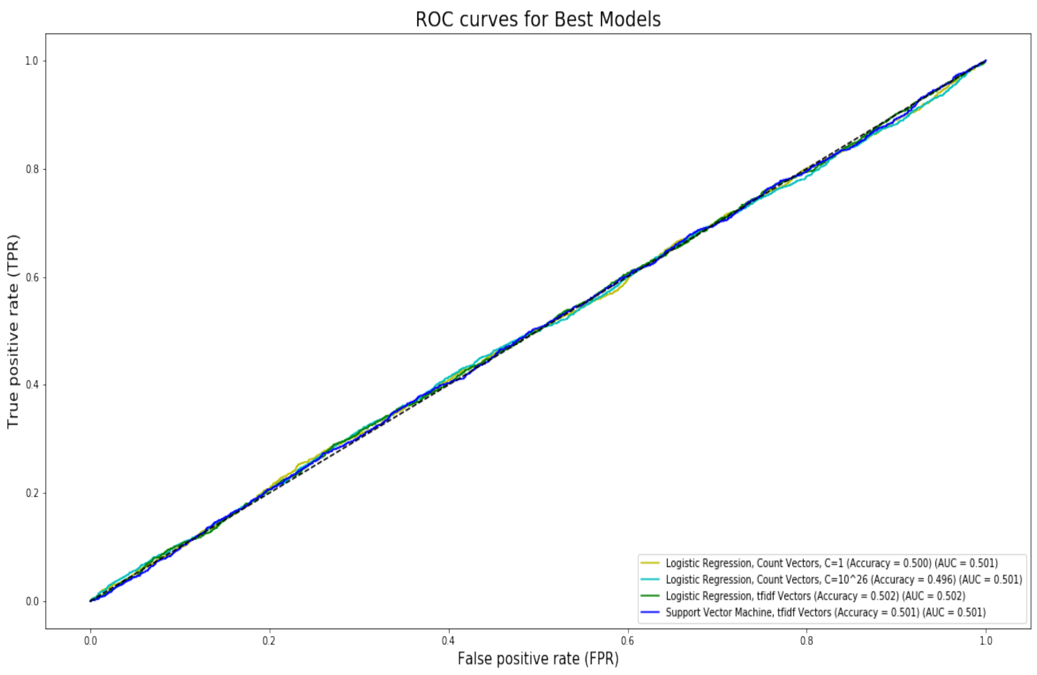


Figure 4.2: LR/SVM Test ROC curves: n-gram range (1, 6)

4.2 Naive Bayes approaches

Three different *Naive Bayes (NB)* approaches were used, namely *Bernoulli*, *Multinomial*, and *Complement*. A brief outline of these methods is given below, followed by a note on model performance. Note that in all the methods below, the classifier assumes probabilistic independence between features.

- **Bernoulli NB**: In this case, the features are binaries of *word/term occurence* as opposed to *word frequencies*. Features are assumed to be drawn from a *Bernoulli* distribution, i.e. the likelihood function is based on the density of a *Bernoulli* distribution. An advantage of these NB classifiers is that they explicitly model the *absence* of terms.

- **Multinomial NB**: *Word/term frequencies* are used as features, and are assumed to be drawn from a *Multinomial* distribution.

- **Complement NB**: The complement NB method was developed *(Rennie et al., 2003)* with the intention of 'correcting' the assumptions of independence that NB methods make. It is said to be suited for imbalanced datasets, as it explicitly calculates the probabilities of a *word/term* **not** occuring in a document. As an example, one would be able say that given that the term *"President Trump"* is present, it is highly likely that document in question does **not** follow a partisan argument.

In order to tune the performance of these models, multiple parameters were tested. This included varying *n-gram* ranges and *vectorization* methods. The results from paramater tuning on the validation set can be found in table 6.2 in Appendix 1. Once the *n-gram* ranges were tuned, n-grams ranging from 1 6 were found to be optimal, and subsequently tested on the test set, along with three different vectorization methods. Apart from the usual *count* and *TF-IDF* vectorizers, a modified version of the *TF-IDF* vectorizer was used- *sublinear TF-IDF*, which considers *logarithmic* term frequencies. Validation and test set ROC curves for all NB models with the different $n-gram$ ranges and vectorization methods used have been shown in Appendix 1- please see figures 6.2 through 6.7.

The model that generalized best turned out to be the *Multinomial Naive Bayes* model with an *n-gram range* of $(1, 6)$ and *count vectorization*, achieving a test set accuracy of 52.10%. The full set of test accuracies has been tabulated below (table 4.2), and the test set ROC curves can be found in Appendix 1- please see figures 6.6 and 6.7.

Model	Vectorization method	Test AUC	Test accuracy
Bernoulli NB	Count	0.507	46.90%
	TF-IDF	0.507	47.40%
	TF-IDF (sublinear)	0.502	47.00%
Multinomial NB	Count	0.531	52.10%
	TF-IDF	0.546	49.45%
	TF-IDF (sublinear)	0.520	48.95%
Complement NB	Count	0.531	52.05%
	TF-IDF	0.546	49.50%
	TF-IDF (sublinear)	0.520	49.00%

Table 4.2: Naive Bayes- Model Performance on test set for n-gram range (1, 6)

4.3 A neural network approach

In order to improve out-of-sample performance, we decided to experiment with a Neural Network approach. The process followed for this approach is described below.

As per the neural network architecture shown in figure 4.3 below, after the embedding layer, a linear layer was used to convert the output to dimension 2. After *softmax*, probabilities of belonging to each of the two classes were obtained.

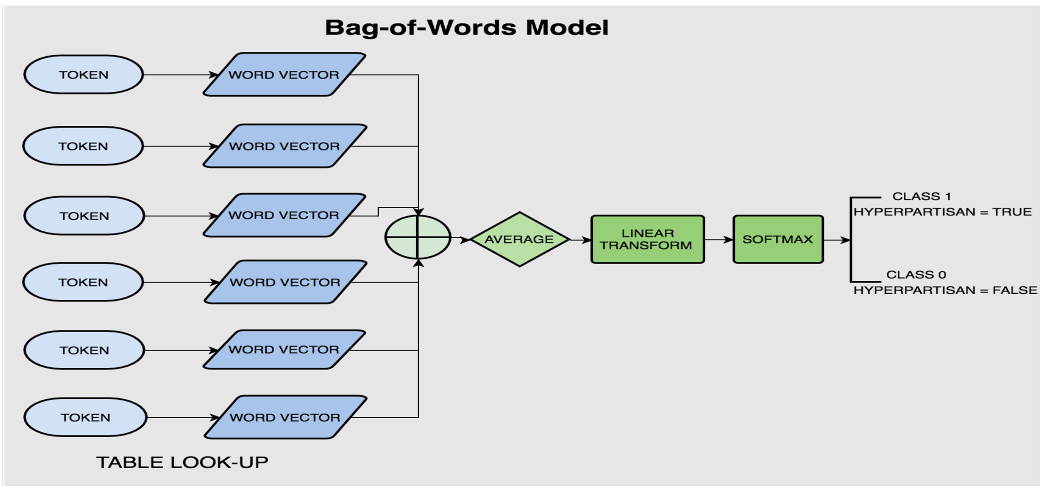


Figure 4.3: A visual representation of the neural network

For the *bag-of-words* (BoW) approach, after obtaining a vector representation of each word in the sentence via an embedding layer, these vectors were summed up element-wise, and represent the encoded information of the input. The method resulted in a validation accuracy of 82% and a test accuracy of 73.2%.

Another approach was also explored- because of the length of the articles, it was not be the best idea to encode all the information in the article as a single vector, summing up the embeddings element-wise. Due to this, we used a *Recurrent Neural Network* (RNN). An RNN is a network which relies not only on the initial input, but also on the output of the previous hidden state. Effectively, it carries a memory portion and articles can be fed to it sequentially. Thus, it takes into account the order of the occuring words, which is important in a task as nuanced as detecting partisanship. Specifically , a bi-directional GRU followed by a fully connected layer was used. Bi-directional usually works better because it concatenates the outputs of two RNNs, one processing the sequence from left to right, and the other one from right to left, which improves the accuracy of the prediction. *Cross-entropy loss* was used as the loss metric, and the *Adam optimization algorithm* was used as the optimizer. The learning rate was set to 0.01, and embedding size to 450. This resulted in a validation accuracy of 87.8%, and a test accuracy of 80.2%.

Additional to the *RNN*, we carried out an *ablation study* on the *embedding size* and *learning rates* in our *BoW* approach. Due to the time constraint, we used the best parameter combination we obtained tuning the previous network in a bidirectional *Gated Recurrent Unit*, using a learning rate of 0.01, an embedding size of 300, and 3 layers followed by 2 fully connected layers.

The results of the ablation study are tabulated below (see table 4.3 below). The choice of learning rates was due to the fact that these have been shown to perform well with *Adam* optimization. Plots showing the variation in accuracy for these parameters can be found in Appendix 2- please see figures 6.8 through 6.11.

Parameter	**Value**	**Test accuracy**
Embedding size	200	60.30%
	300	70.50%
Learning rate	0.001	73.20%
	0.0003	64.30%

Table 4.3: Ablation study: results

As seen above, increasing the *embedding size* significantly improved test performance. This is because in our BoW model, the embedding vectors of each input are piece-wise summed before being fed into a linear layer. Thus, the summation of the vectors encodes the semantics of the article. Adding dimensionality will therefore encode more information in the summation.

Meanwhile, increasing the learning rate also led to better performance. However, as seen in figures 6.8 through 6.11, training with a higher learning rate led to higher instability during training, as the optimizer first overshot the minimum in the loss function manifold (as shown by the drastic increase and decrease in validation set preformance in the first epoch).

CHAPTER 5 | MODEL SELECTION AND DEPLOYMENT

5.1 A note on model selection

The maximum test accuracy obtained across our various models was 80.2%, using a recurrent neural network with a learning rate of 0.01 and an embedding size of 450. While there was a significant drop in accuracy from the validation set to the test set, it was not as drastic as that seen with the *logistic regression*, *SVM*, *Naive Bayes*, or untuned neural network models. The test set consists of articles from publishers that are not present in the training set, and the distribution of data (i.e. vocabulary) in the test set seems to be very different from that in the training set.

Further work can be done to improve our classifiers' performance, including, but not limited to:

- Training on a larger number of articles to ensure a wider vocabulary
- Training on more embedding sizes
- Attempting linear annealing for learning rate
- Attempting to incorporate various n-grams along with uni-grams in the neural network

5.2 Code

All code for this project was written in Python, and can be found on GitHub, in this repository.

CHAPTER 6 | APPENDIX 1: TABLES AND PLOTS

Model	**Regularization weight**	**Validation AUC**
LR (count vec.)	10^{-30}	0.500
	10^{-15}	0.812
	10^{-05}	0.876
	1	0.925
	10^{+5}	0.921
	10^{+15}	0.919
	10^{+30}	0.919
LR (TF-IDF vec.)	10^{-30}	0.500
	10^{-15}	0.500
	10^{-05}	0.874
	1	0.894
	10^{+5}	912
	10^{+15}	0.910
	10^{+30}	0.909
SVM (count vec.)	10^{-30}	0.864
	10^{-15}	0.864
	10^{-05}	0.864
	1	0.864
	10^{+5}	0.864
	10^{+15}	0.864
	10^{+30}	0.864
SVM (TF-IDF vec.)	10^{-30}	0.916
	10^{-15}	0.916
	10^{-05}	0.916
	1	0.916
	10^{+5}	0.916
	10^{+15}	0.916
	10^{+30}	0.916

Table 6.1: LR/SVM: Parameter tuning results

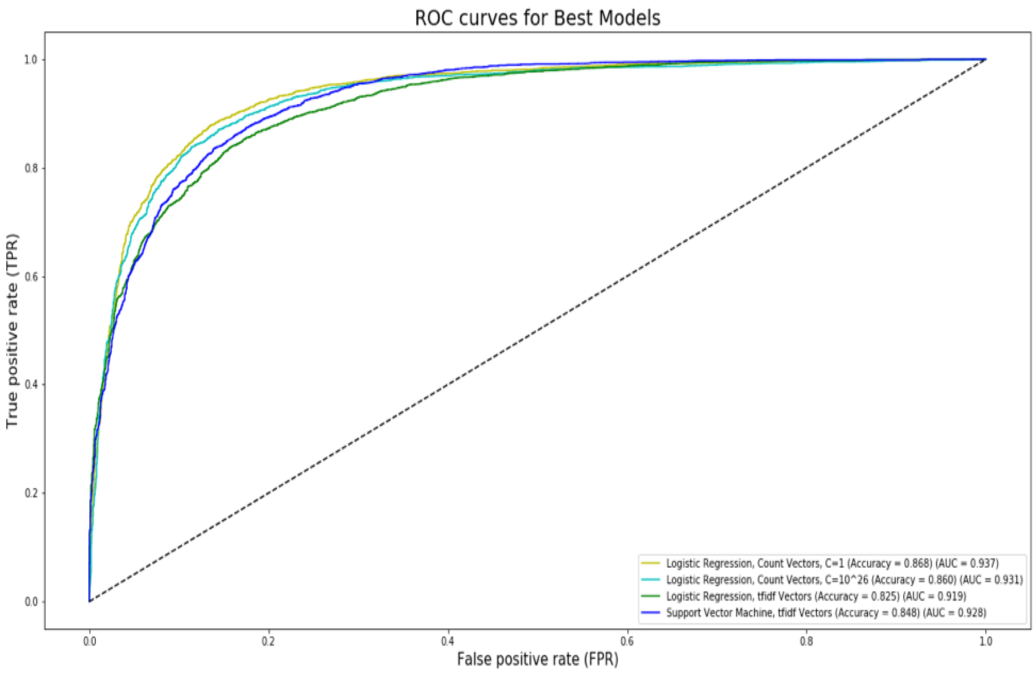


Figure 6.1: LR/SVM Validation ROC curves: n-gram range (1, 4)

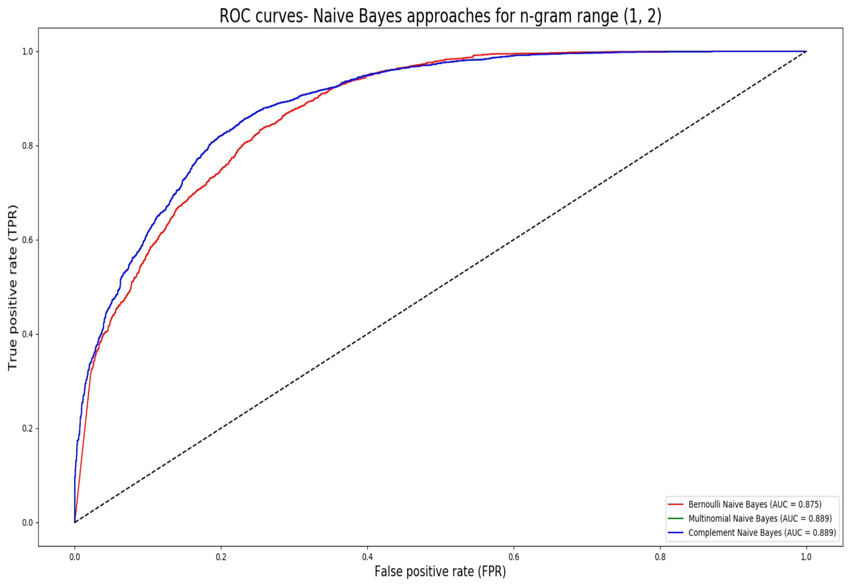


Figure 6.2: Naive Bayes ROC curves: n-gram range (1, 2)

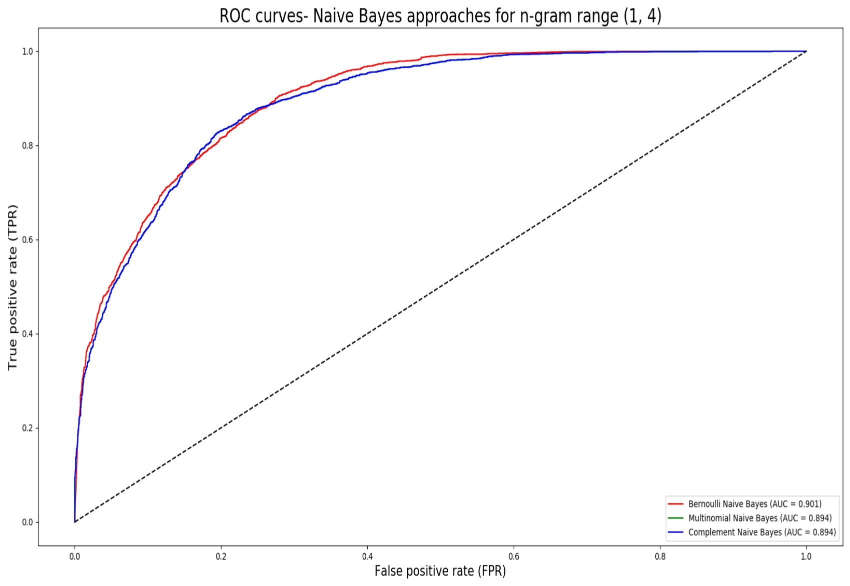


Figure 6.3: Naive Bayes ROC curves: n-gram range (1, 4)

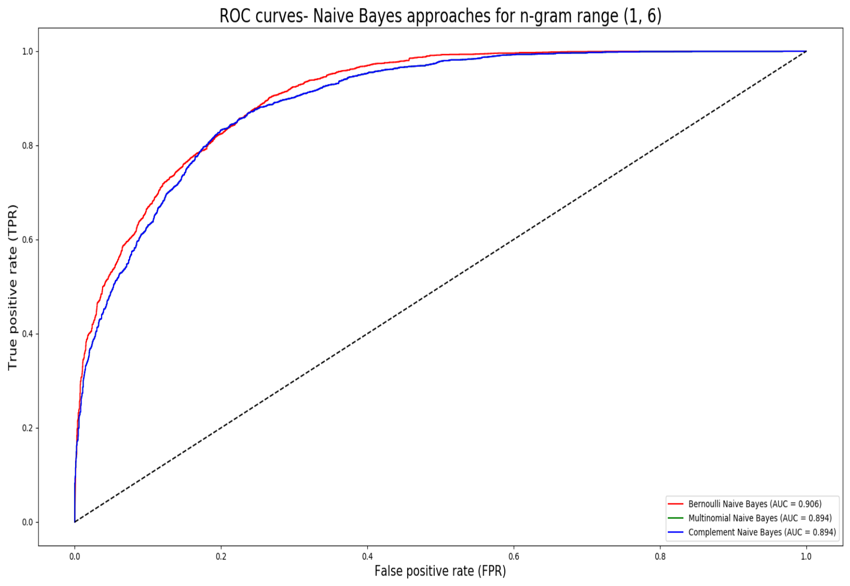


Figure 6.4: Naive Bayes ROC curves: n-gram range (1, 6)

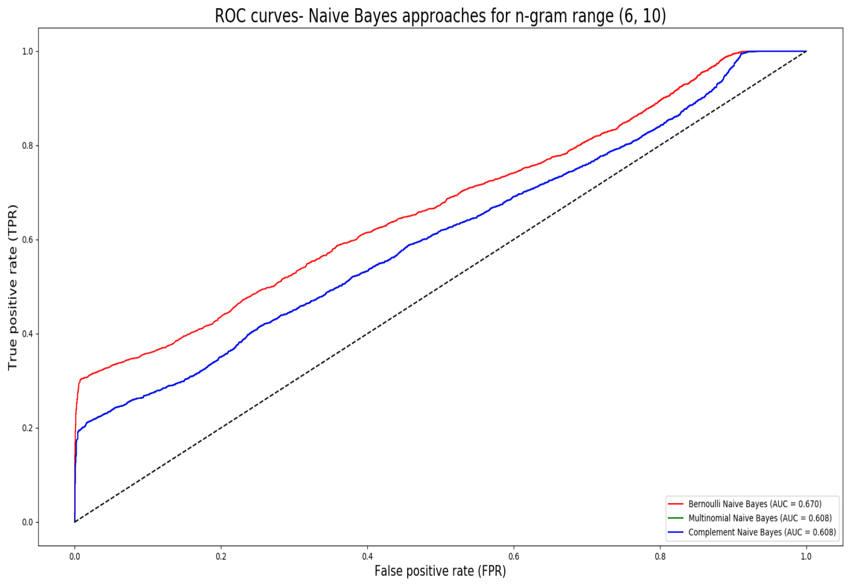


Figure 6.5: Naive Bayes ROC curves: n-gram range (6, 10)

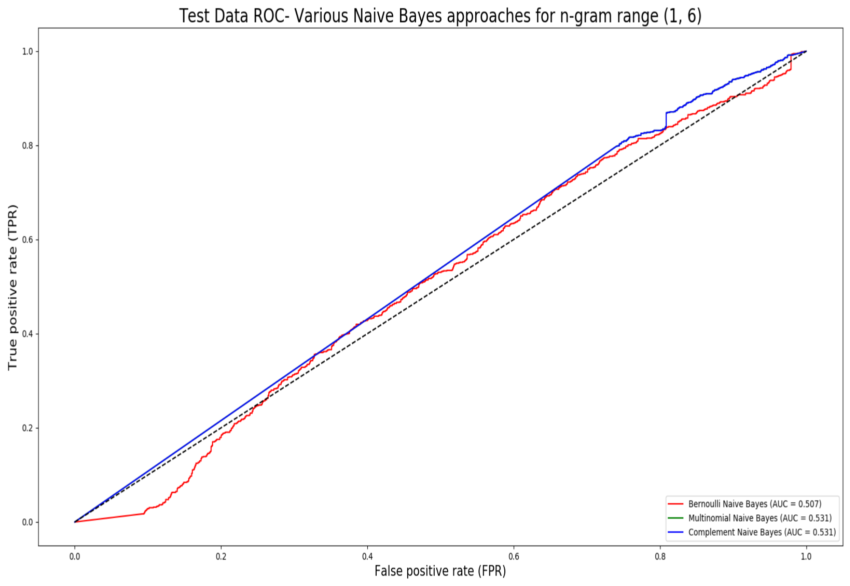


Figure 6.6: Naive Bayes ROC curves- test data (count vectorization)

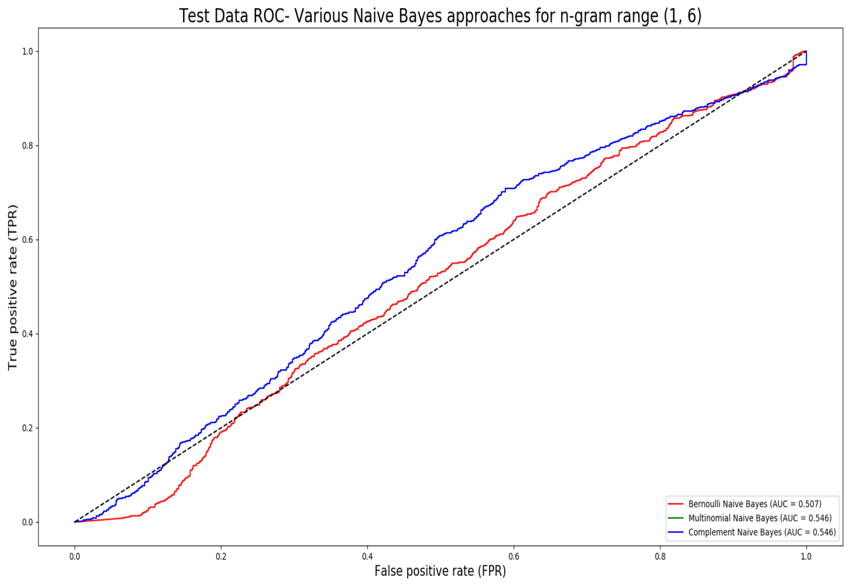


Figure 6.7: Naive Bayes ROC curves- test data (TF-IDF vectorization)

Model	**n-gram range**	**Validation AUC**	**Validation accuracy**
Bernoulli NB	(1, 2)	0.875	66.82%
	(1, 4)	0.901	58.23%
	(1, 6)	0.906	55.45%
	(6, 10)	0.670	54.08%
Multinomial NB	(1, 2)	0.889	66.82%
	(1, 4)	0.894	58.23%
	(1, 6)	0.894	55.45%
	(6, 10)	0.608	54.08%
Complement NB	(1, 2)	0.889	78.52%
	(1, 4)	0.894	78.82%
	(1, 6)	0.894	78.83%
	(6, 10)	0.608	57.83%

Table 6.2: Naive Bayes- Model Performance (validation set)

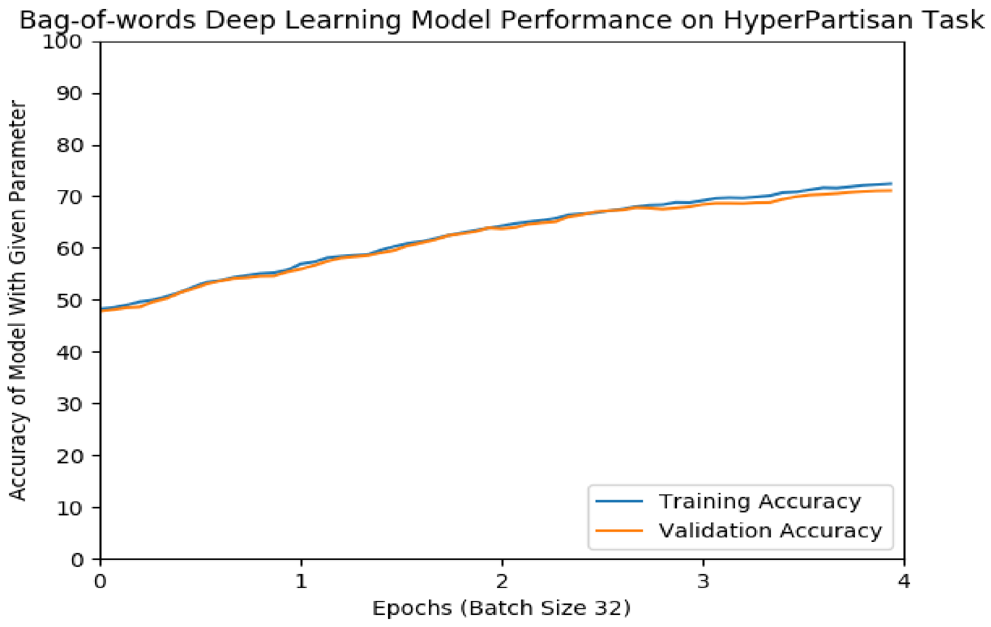


Figure 6.8: Ablation study: accuracies for embedding size = 200

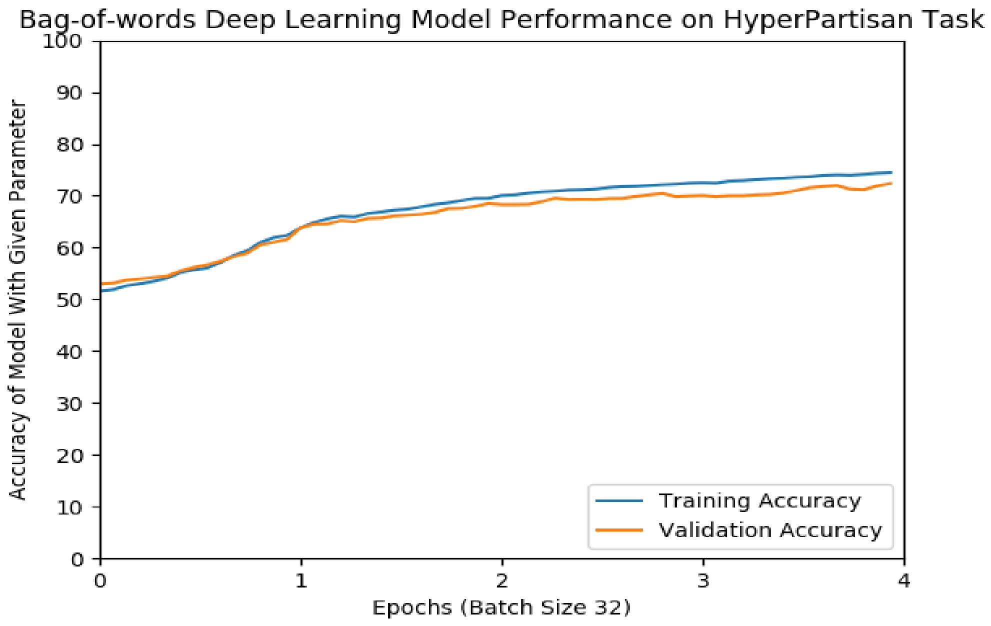


Figure 6.9: Ablation study: accuracies for embedding size = 300

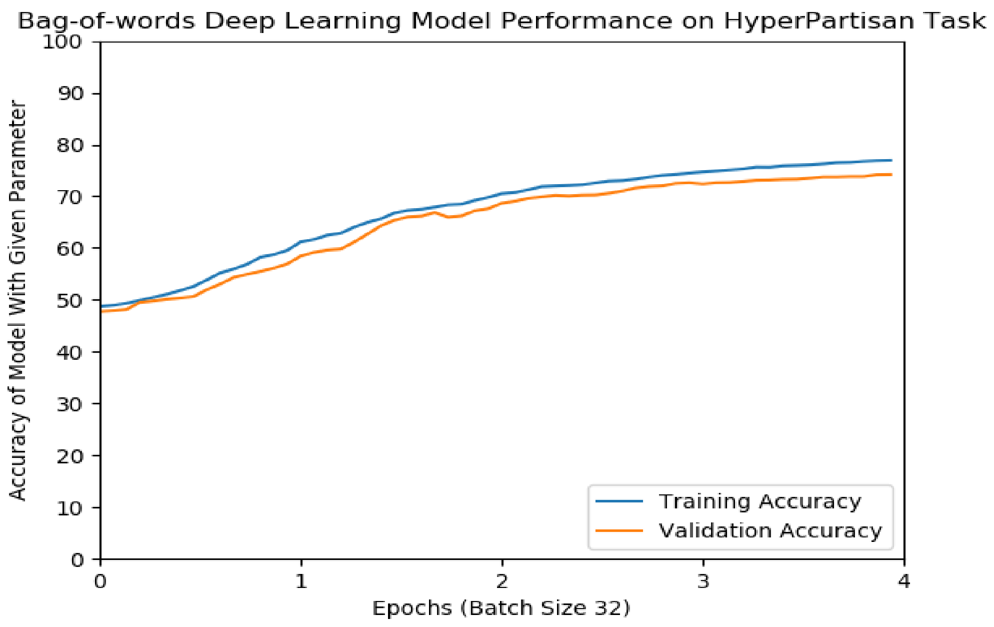


Figure 6.10: Ablation study: accuracies for embedding size = 300 and learning rate = 0.0003

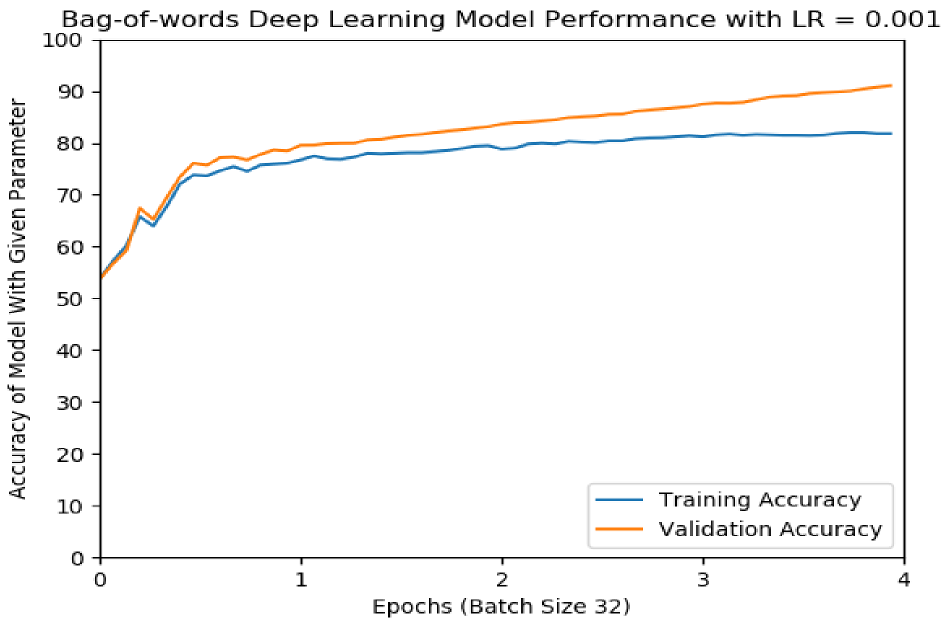


Figure 6.11: Ablation study: accuracies for embedding size = 300 and learning rate = 0.001

CHAPTER 7 | APPENDIX 2: CONTRIBUTIONS

The workload was divided between team members as follows:

- **Yash**: Initial data loading and transformation; data pre-processing; design, implementation, and write-up for Naive Bayes approaches; report editing
- **Yada**: Pre-processing for the neural network approach; design, implementation, and write-up for the neural network approach; model evaluation and selection approach
- **Aja**: Logistic regression and SVM approaches: design, implementation, and write-up
- **Jiayi**: data pre-processing (neural network approach); design, implementation, and write-up for the neural network approach; RNN and GRU evaluation