IDS 703: Final Project

Kashaf Ali, Elisa Chen and Pragya Raghuvanshi

1. Motivation & Problem Statement

According to Pew Research Center, roughly one-quarter of American adults use Twitter, and often it is used to share their views about politics and political issues[1]. Twitter can have a huge impact on the political sphere in the United States, and it can be helpful in determining political biases in online discussions related to important socio-political topics. Previous research also suggests that the advent of social media platforms like Twitter and the increase in their use has generated huge amounts of data containing unstructured text in the form of public posts like tweets, messages, and blogs. As people use these platforms to express their political interests, this data can be used to analyze people's opinion about different political parties, and even use it to predict future election trends[2].

In this project, we are solving a **text/document classification problem** and using the political orientation of certain tweets to identify the presence of Democratic vs. Republican content on Twitter and identify the issues each political party cares the most about. More specifically, we are creating a generative and a discriminative model to predict the political orientation ('Democratic' or 'Republican') for a specific tweet. In terms of a generative model, we trained a naive bayes classifier and used it to generate synthetic data that we will test with both our generative model, and our bidirectional LSTM model to compare performance.

2. Methodology

2.1. Data

2.1.1. Introduction

The original dataset used in this analysis consisted of senator tweets derived from Hugging Face[3]. The dataset has approximately 99,000 observations/rows and 6 columns. The columns of interest for our analysis are with tweets content, and class labels (Democrat and Republican). This dataset was cleaned and processed, and information relevant to our analysis was retained. The dataset was split into training and testing in the ratio of 80:20.

[1]https://www.pewresearch.org/politics/2022/06/16/politics-on-twitter-one-third-of-tweets-from-u-s-adults-are-political/
[2] https://www.sciencedirect.com/science/article/pii/S1877050920306669
[3]https://huggingface.co/datasets/m-newhauser/senator-tweets

2.1.2. Data Cleaning and Preprocessing

The raw tweets prior to preprocessing were highly unstructured and contained redundant and problematic information. Hence, dataset cleaning and preprocessing are necessary steps to perform on the data prior to its analysis. Foremost, the original dataset contains certain columns that are not required in our analysis. So the columns were dropped and only the one containing the tweets data and the labels of the classes, that is Democrat or Republican, were retained. The following initial data cleaning process was performed on the tweets:

- Dropping unnecessary columns and retaining the tweets and class labels
- Replacing NaN values with empty string
- Removing all URLs from the tweets
- Removing mentions and hashtags
- Converting all tweets to lowercase for better generalization
- Removing punctuations, emoticons, non-alphanumeric characters and numbers reducing the unnecessary noise
- Removing words with less than two or less than two characters as they usually do not hold any important information.
- Stop words that are commonly occurring words and do not have any useful information are removed using the Stop Words dictionary from NLTK.

Post cleaning for unnecessary noise, the next step is to perform certain data pre processing steps crucial to our analysis:

- *Tokenization* : Tokenization is a mandatory aspect of working with textual data. It is a process of separating the tweets into smaller tokens that are more understandable. It can be words, sub words or characters. Here we segment our tweets in word tokens using word_tokenize from NLTK library.
- *Spelling Correction*: Created a dictionary for all misspelled words and used textblob to perform a spelling correction on them and replace those words by their correct spelling.
- *Stemming*: This is the process of eliminating affixes (circumfixes, suffixes, prefixes, infixes) from a word in order to obtain a word stem. For instance, the stem of these three words, development, developing, develops, is “develop”. We used Porter Stemmer from NLTK for this purpose.
- *Lemmatization*: Lemmatization is the process of converting the words into their root words called lemma. For example , it will group the word “good” together with “better” and “best”.

2.1.3. Exploratory Data Analysis

A quick look into the summary statistics of our data reveals that prior to the cleaning process the mean length of tweets is around 215 characters which shrinks to a length of 173 after the cleaning, for our training dataset. Similarly for the test dataset the length of tweet reduces from 215 to 172 characters.

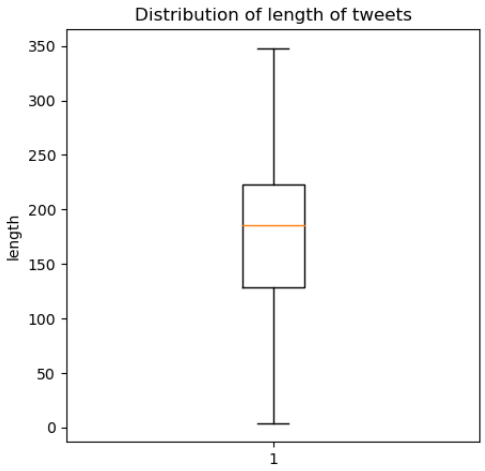

Figure 1: Boxplot for length of tweets in training dataset

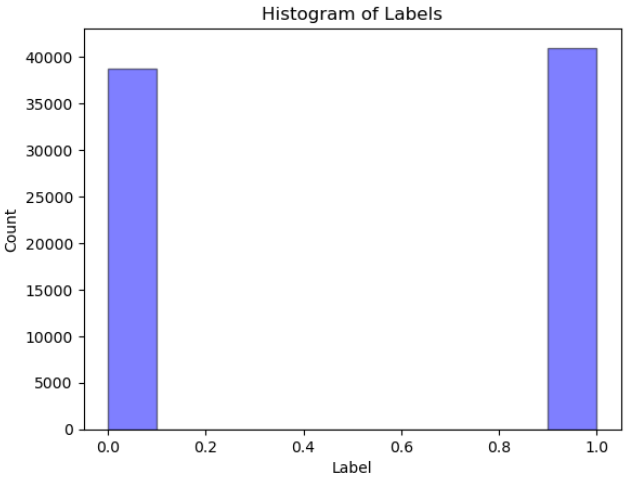

Figure 2: Histogram for checking balancing in classes

The boxplot above for our training dataset suggests no noticeable skewness in our training data, as the median length of tweets is around 180, which is almost equal to the mean. For proper training of our model, it is imperative to have a balance between our two classes in the training dataset. From the histogram shown above we can check the balancing of the two classes, Democrat and Republican. We can clearly observe an almost equal count of 0 (Republican) and 1 (Democrat) labels.

2.2. Models

For the classification of tweets in the two class labels of Democrat and Republican, we built two models and conducted training and testing on them.

1. *Generative mod*el: We used the naive bayes classifier as our generative model as it learns the joint probability distribution p(x,y) and can use conditional probability to predict the class of a tweet.
2. *Discriminative model*: We used a bidirectional LSTM as our discriminative model as it consists of two LSTMs, which take input from both forward and backward directions, and essentially increases the amount of information available to the network to make predictions.

3. Generative Model

3.1. Description

We employed the naive bayes classifier as a generative model to classify tweets based on political affiliation. A naive bayes classifier is a machine learning model that assigns classes from some finite set to various problem instances, which are represented as vectors of feature values[4]. This model is based on the bayes' theorem, which assumes that all features are independent of each other, or in other words, the presence of a particular feature in a class is unrelated to the presence of any other feature.

The naive bayes model is commonly used for text classification problems, where a text document is represented as if it were a bag-of-words, which is an unordered set of words with their frequencies in the document; in this case, the position of the word does not matter. To predict the most probable class given some document, we choose the class which has the highest product of two probabilities: the prior probability of the class and the likelihood of the document.

We trained the naive bayes classifier using the Scikit-learn package based on the following steps:

1. *Data Pre-processing:* we carried out similar data pre-processing as described above for the naive bayes model. However, we did not add in the spelling check function in our cleaning process for this model as excluding it resulted in a slightly higher model accuracy, which might indicate that there might be some instances where the spelling correction mappings might have resulted in replacing legitimate words with incorrect ones.
2. *Feature Engineering:* this step involved taking features, which are important pieces of information, from the text and giving it to the model to make predictions. In this case, we created a bag-of-words consisting of unigrams, bigrams, trigrams and 4-grams from our tweets data and calculated their frequencies, which are the counts of each token's occurrence in our tweets data. Then, we used these frequencies to calculate the probabilities that the given token in our training data belongs to a specific class (Republican and Democrat). We calculated these probabilities for all features and used them to calculate the probability of a tweet belonging to one of the defined classes. It is important to note that we used all four different kinds of n-grams to create our bag-of-words because previous research showed that using different kinds of n-grams together leads to better accuracy and gains in text classification problems as they are more likely to capture modified verbs and nouns than just unigrams[5]. Moreover, we also

[4] https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[5] https://aclanthology.org/P12-2018/

used the CountVectorizer to vectorize our tweets in an encoding vector that returned the length of the entire vocabulary and integer counts for the number of times each token occurs in the tweets. We set max_df as 0.90 and min_df as 2 as parameters in our CountVectorizer to ensure that terms that appear in more than 90% of the documents are ignored and terms that appear in less than 2 documents are ignored as well.

3. *Train and Test Splitting:* We used the train_test_split module of scikit-learn to divide the dataset into train and test sets in the 80:20 ratio to first train the model and then measure its performance on the test set.
4. Multinomial Naive Bayes Classifier - As our dataset has a multinomial distribution, we used the multinomial naive bayes classifier, which is generally used for text classification problems and is appropriate for features that represent count or count rates. In this model, each event would represent the occurrence of a token in a document. We also created a function for finding the best alpha for smoothing the model, and found that the alpha 0.2 led to the highest accuracy score for our model. We also used the Grid Search to further validate this.

3.2. Performance with Real Test Data

We tested our model using the training set to understand how well our naive bayes model was able to classify tweets into the Republican and Democrat classes. The confusion matrix below shows that the true positives and true negatives for both our classes, Republicans (0) and Democrats (1) are higher than the false positives and false negatives. Moreover, we also used the following metrics to understand our model's performance.

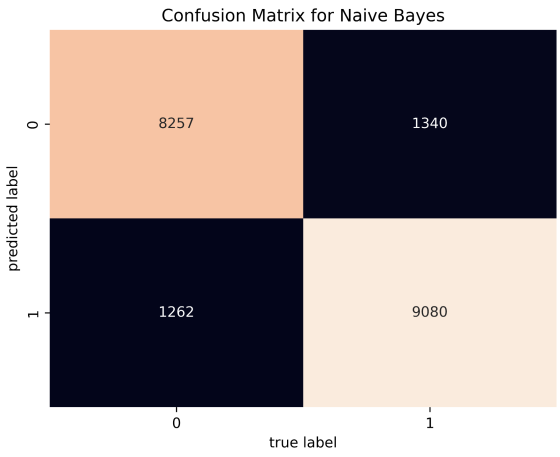

Figure 3: Confusion Matrix for Real Data

Metrics	Score
Accuracy	0.8695

F1-Score	0.8695
Area under ROC	0.8692

Table 1: Performance metrics for Real Data

4. Discriminative Model

4.1. Description

A bidirectional LSTM (biLSTM) model was chosen to perform the text classification task for the following reasons: 1) LSTMs are great at retaining relevant information while addressing the exploding gradient problem, 2) Bi-directional structures are designed to infer information both from the past and future. We did also consider using transformers, which are great at capturing long-range dependencies, but decided that LSTMs are more appropriate for our purpose given the shorter lengths of tweets and the desire for a simpler model. The architecture of the LSTM model is shown below:

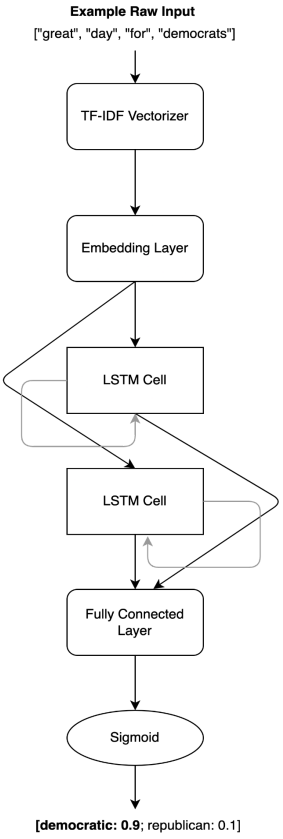

Figure 4: BiLSTM Structure

As observed, first the vectorized input is passed through an embedding layer to improve the representation of the tweet. The size of each embedded vector is 200, which is a fairly standard size for text classification tasks. The LSTM cell would take in the embedded vector, and the

outputs from the hidden layer (which are initialized to a 0 vector), before passing through the final fully-connected layer. Prior to passing through the final layer, the results are fed through the dropout layer to improve the regularization of parameters, which will randomly zero some of the elements of the input tensor with a probability of 30% using samples from a Bernoulli distribution. Lastly, the outputs are passed through a fully-connected layer that consolidates the outputs from the bidirectional layers and is passed through a sigmoid activation function.

4.2. Performance with Real Test Data

In comparison to the generative model, The biLSTM model performs relatively poorly on training and testing data achieving a testing accuracy of ~51% and a F-1 score of ~0.45 on average for 10 epochs. Please see the below graphs more details:

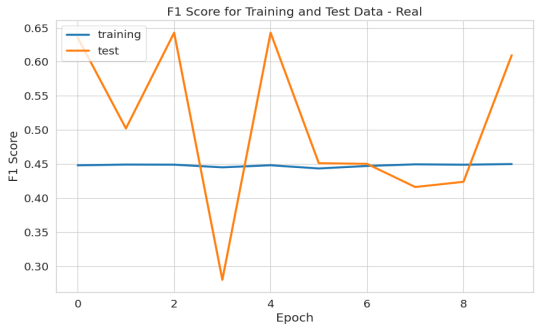

Figure 5: F1 Score for Real Training and Test Data

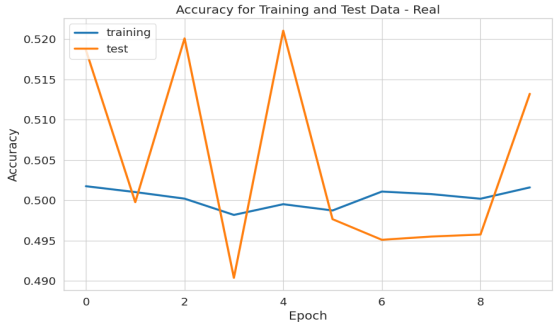

Figure 6: Accuracy for Real Training and Test Data

Based on the confusion matrix generated on 100 tweets randomly sampled from the real dataset, we observe that the model is more likely to predict democratic (label: 1) tweets incorrectly. The sample data was balanced between democratic and republican tweets so perhaps this indicates that republican tweets are more identifiable compared to democratic tweets.

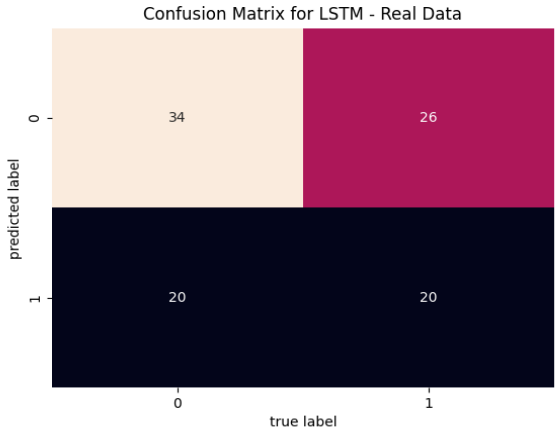

Figure 7: Confusion Matrix for LSTM Real Data

5. Synthetic Data Generation

The generative model is used to create synthetic data and testing is performed on both models. As a generative model essentially includes the distribution of the data itself, it can be used to produce synthetic data that mimics the original data by drawing from the distribution. A generative model learns the joint probability p(x,y) and estimates the posterior probability of the classes (Democrat and Republican). We use Naive Bayes as our generative model where the model makes this "naive" assumption that features are independent of each other given the class. As we try to emulate synthetic textual data from the probability distribution, it is mostly certain that the text would not follow the usual grammar, since language essentially depends on the sequencing and context of the words. We generate synthetic data equivalent to the sample size of the real data divided equally between the two classes with each tweet length ranging anywhere between 40 to 60 characters.

5.1. Performance

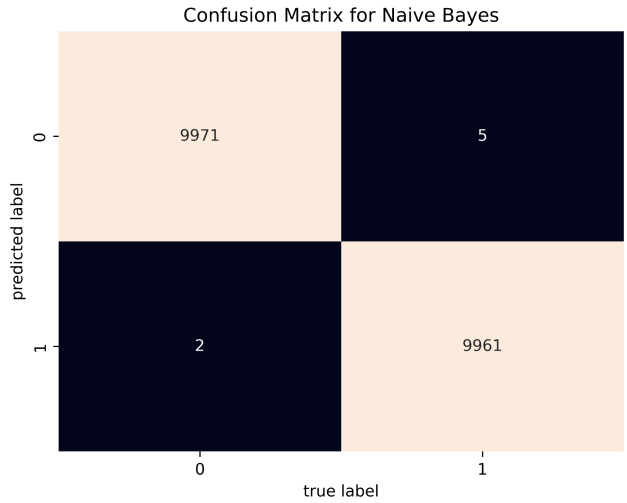

Figure 8: Confusion Matrix for Naive Bayes

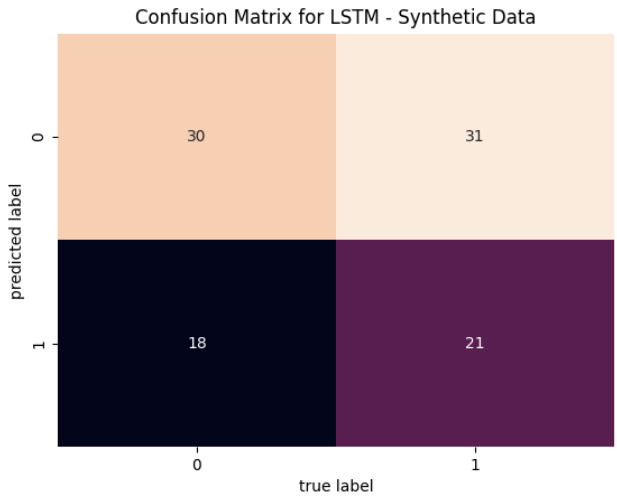

Figure 9: Confusion Matrix for biLSTM

With the synthetic data, the naive bayes give an accuracy of around 99.6%. As the data was itself generated by the model mimicking the original probability distribution, it is likely that the generative model will predict them with high accuracy.

Metrics	Naive Bayes Score	BiLSTM Score
Accuracy	0.9996	0.50
F1-Score	0.9996	0.40

Table 2: Performance metrics for Synthetic Data

With bidirectional LSTM, the accuracy and the F-1 score for synthetic data were quite similar to the performance of real data. From the confusion matrix shown above, generated again on 100 randomly sampled tweets, we can observe that the model is more likely to predict democratic (label: 1) tweets incorrectly. This reinforces our earlier findings obtained from the real data that republican tweets are more easily identifiable when compared to democratic ones. The training accuracy was on average ~50% and the F-1 score was 0.40 on average for 10 epochs. The testing accuracy was ~50% and the F-1 score was 0.53 on average for 10 epochs. Please see the below graphs for more details:

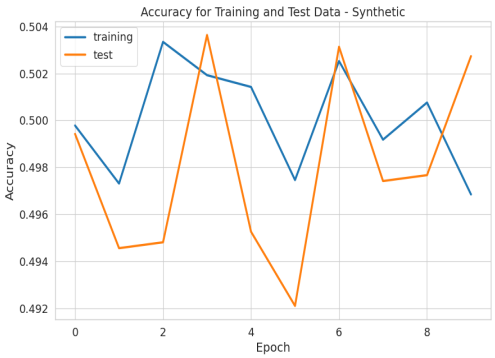

Figure 10: Accuracy for Synthetic Training and Test Data

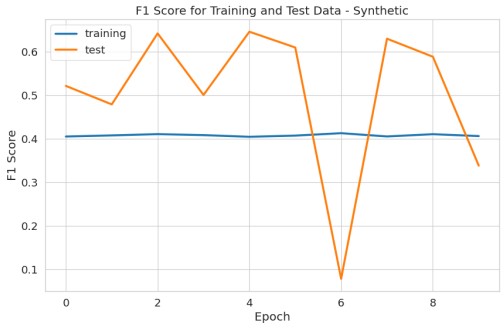

Figure 11: F1 Score for Synthetic Training and Test Data

5.1.1. Limitations

The synthetic dataset generated by Naive Bayes model does not make much sense grammatically and is essentially a bag-of-words model picked on the probability distribution of the original data Hence, the biLSTM, that learns the features through a training process on the original dataset having mostly correct grammatical sequence, is bound to have lower accuracy for synthetic data generated by Naive Bayes. There are a number of reasons why the biLSTM achieves a lower accuracy and F-1 score than the generative model:

1. By using TF-IDF to vectorize the tweets, the model disregards the ordering of words, which makes it hard for it to learn the context and the semantic relationship between words.

2. The learning rate of the model was 0.01 due to performance constraints. Lowering the learning rate could have improved the model metrics.
3. The model was trained on <100k samples, which is not enough to achieve high accuracy. Perhaps the model metrics could have been improved by training the model on a larger and more diverse dataset.

6. Conclusions

The generative model performed relatively better on the text classification task compared to the discriminative model on both real and synthetic data. The generative model is faster to train on a small dataset and is more intuitive to interpret than the discriminative model. However, the generative model relies on the assumption that the data is sampled from a particular probability distribution, which is rarely true in reality. Therefore, if you generate synthetic data that is sampled from the assumed distribution, the generative model is expected to achieve a higher model accuracy and F-1 score compared to if the model was trained and tested on real data.

The discriminative model will learn the characteristics of the training dataset through backpropagation, and given that the test data is similar to the training data, the discriminative model should perform well. However, this is computationally very intensive and will require a lot of data, computing power, and time to achieve good results, which was not feasible for our analysis. The discriminative model took ~6h to train on ~80k tweets.