MIE1624 Assignment 2 Report

Data Cleaning

The following steps were done to clean the raw tweet data:

- Removed all URLs and HTML tags, and replaced all HTML character codes with ASCII equivalent
- Removed retweet segment from retweets as well as user mentions (@)
- Removed all numbers and special characters, and changed all characters to lowercase
- Used the nltk stop words library to remove stop words

The first 3 raw tweets from *sentiment_analysis* alongside their cleaned version can be seen below:

text	clean text
b"@RosieBarton So instead of your suggestion, ...	instead suggestion agree canadian women trust ...
b"#AllWomanSpacewalk it's real!\n@Space_Statio...	allwomanspacewalk realn netobicokenorth cityof...
b"#Brantford It's going to cost YOU $94 BILLIO...	brantford going cost billion next years ask ju...

Exploratory Analysis

To determine the political party affiliation of tweets in the *Canadian_elections_2019* dataset, I implemented the function politicalParty():

- The function takes as input a list of n tweets, and outputs a list of size n with the values in the list being the political party affiliation for each tweet (liberal, conservative, NDP, mutual, or none)
- If a tweet is predicted to belong to more than one political party, its value in the will be "mutual"
- If a tweet is predicted to have no political affiliation, its value will be "none".
- the function works by checking if a tweet contains at least one word from a list of relevant words for each political party

The lists of relevant political party words I used to categorize the tweets can be seen below:

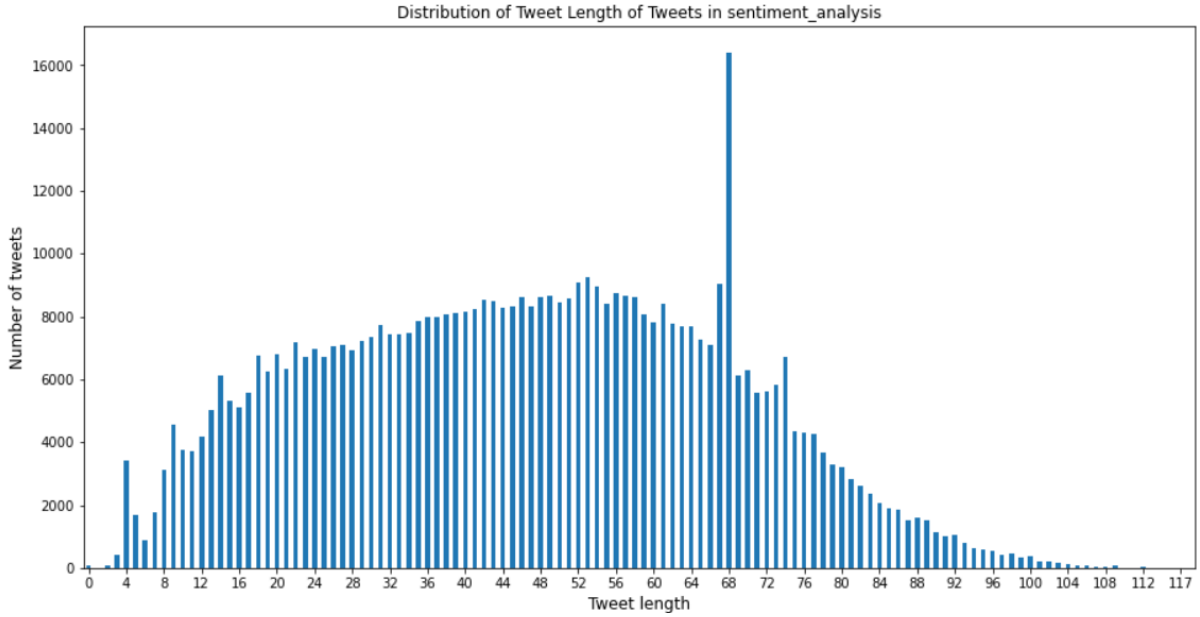
```
# list of common words for Liberal Party
liberal_words = ['liberal', 'lpc', 'justin', 'trudeau']
# list of common words for Conservative Party
conservative_words = ['conservative', 'tory', 'tories', 'cpc', 'andrew', 'scheer', ]
# list of common words for New Democratic Party
ndp_words = ['new democratic party', 'ndp', 'jagmeet', 'singh']
```

The distribution of the political affiliation of the tweets can be see below:

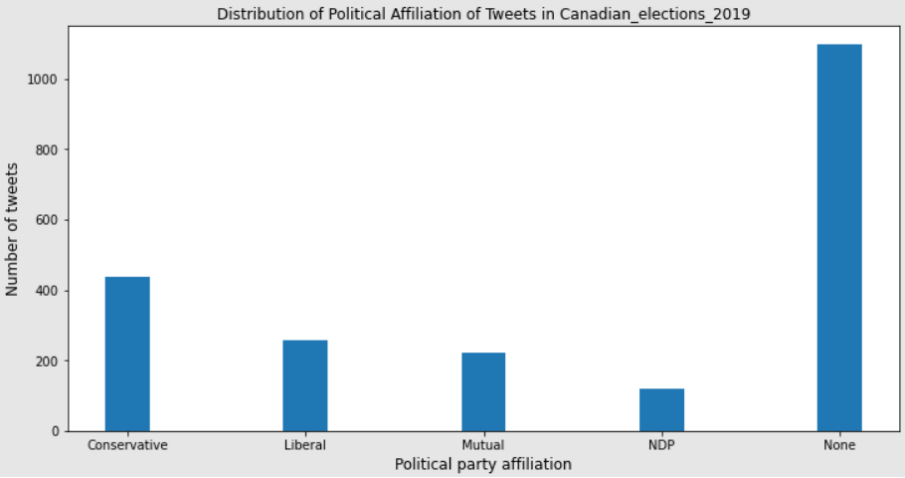
```
Political Party
Conservative     438
Liberal          256
Mutual           221
NDP              121
None            1097
```

Out of the three political parties, the Conservative Party is affiliated with the largest number of tweets, followed by the Liberal Party. The New Democratic Party is affiliated with the least amount of tweets out of the three parties.

I plotted two graphical figures to visualize some of the important aspects of the two tweet datasets. The first figure is a bar plot of the distribution of tweet length in *sentiment_analysis*:



The second figure is a bar plot of the distribution of political affiliation of tweets in *Canadian_elections_2019*:



Model Preparation

I used both the bag of words method and TF-IDF method for converting the tweets into numerical features. I created two functions, bagOfWords() and tf_idf(). Both functions take as input a list of cleaned tweets and outputs the sparse matrix containing the extracted numerical features of the tweets. For both functions, I set the max_features parameter to 2000 in order to speed up the runtime of the functions.

I implemented 7 different functions for 7 different classification models. They are, respectively: logistic regression, k-nn, multinomial Na ïve Bayes, linear SVM, decision tree, random forest, and gradient-boosted trees. The functions I implemented have the following characteristics:

- All functions use *sklearn* as the model, except for gradient-boosted trees. The gradient-boosted trees function uses model from *XGBoost*
- All functions take as input the training and test sets of features array and target, and outputs the model accuracy
- For Naïve Bayes, I chose to use the multinomial option since it generally has better performance compared to Gaussian and Bernoulli in NLP classification problems
- For SVM, I chose to use the linear option for faster convergence
- The logistic regression function takes the hyperparameter C (inverse of regularization strength) as an optional input (default value is set to 1)
- The multinomial Naïve Bayes function takes the hyperparameter alpha (Laplace smoothing) as an optional input (default value is set to 1)
- The gradient-boosted trees function takes the hyperparameter learning_rate (learning rate) as an optional input (default value is set to 0.3)

A 70-30 train-test split was done on the cleaned tweets of the sentiment_analysis dataset using train_test_split() function from *sklearn*.

Model Implementation and Tuning

For predicting tweet sentiment, each of the 14 models (7 classification algorithms, 2 feature types) were trained on the training set of the tweets from *sentiment_analysis*. After applying the models to the test set, the following accuracy table was obtained:

```
Accuracy of models:
```

	Bag of words	TF-IDF
Logistic regression	0.9525	0.9522
k-NN	0.9194	0.8478
Naive Bayes	0.9259	0.9556
SVM	0.9517	0.9523
Decision tree	0.9322	0.9344
Random forest	0.9435	0.9462
Gradient-boosted trees	0.9246	0.9233

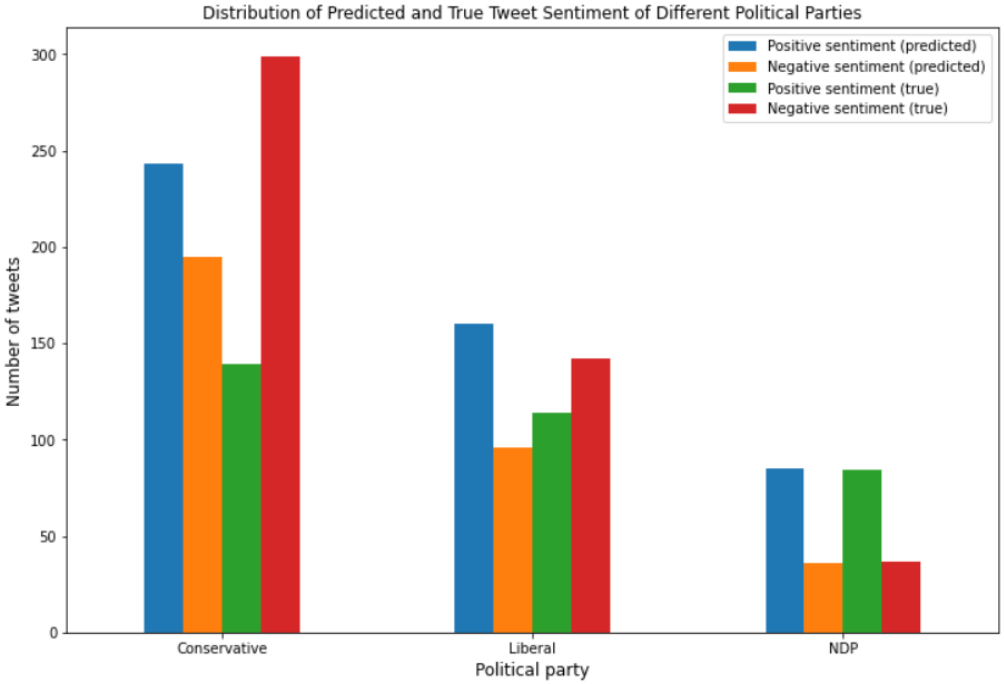
```
best model: Naive Bayes using TF-IDF features
accuracy of best model: 95.56 %
```

The best-performing model, multinomial Naïve Bayes using TF-IDF features, was then tested on the tweets from the *Canadian_elections_2019* dataset.

```
Accuracy of best model when used to predict sentiment in Canadian_elections_2019 dataset: 51.00796999531176 %
```

The best model, multinomial Naive Bayes using TF-IDF features, only has an accuracy of 51% when used to predict sentiments in the elections dataset. This means that the model predicted the correct sentiment on just over half the tweets in the elections dataset. This same model has an accuracy of 95.25% when used to predict sentiment in *sentiment_analysis* dataset, meaning that there was a big drop in accuracy when the test set was switched from *sentiment_analysis* to the elections dataset.

To visualize the predicted sentiment and true sentiment of the three political parties, I created the following bar plot of the distribution of true and predicted tweet sentiment.



I believe that natural language processing is very useful for political parties during election campaigns. First of all, political parties can use NLP to predict the public's overall sentiment of their political opponents (opposing parties). From this, the political party can get a general idea of where they stand in terms of the general public (e.g. if a party's sentiment is better than the sentiment of the opposition, it means that the party is likely to win the election). Political parties can also use NLP analytics obtained from tweets to make important election campaign decisions. For example, since twitter users can be traced by the geographical location (hometown) that they set in their profile, political parties can use these analytics to determine which regions/ geographical areas have an overall negative sentiment of the party. The political party can then decide to invest more time and resources to hold additional campaigns in these regions to try and improve this sentiment. This is especially important in the "swing states" during US elections, since these states are often very hard to predict.

For predicting the reason for negative sentiment, I first performed a 70-30 train-test split on the tweets in the Canadian_elections_2019 dataset. I then chose 3 classification models – logistic regression, multinomial Naïve Bayes, and gradient-boosted trees – to train on the training set using TF-IDF features. I performed hyperparameter tuning for each model (C for logistic regression, alpha for multinomial Naïve Bayes, and learning_rate for gradient-boosted trees) using 5-fold cross validation. The results can be seen below:

```
logistic regression:
C = 0.01 , accuracy: 36.22 %
C = 0.1 , accuracy: 37.64 %
C = 1 , accuracy: 45.89 %
C = 1000 , accuracy: 51.15 %
C = 10000 , accuracy: 50.72 %
```

```
Naive Bayes:
alpha = 0.0001 , accuracy: 44.61 %
alpha = 0.001 , accuracy: 44.6 %
alpha = 0.01 , accuracy: 47.02 %
alpha = 0.1 , accuracy: 47.45 %
alpha = 1 , accuracy: 45.6 %
```

```
gradient-boosted trees:
learning rate = 0.001 , accuracy: 49.57 %
learning rate = 0.01 , accuracy: 52.56 %
learning rate = 0.1 , accuracy: 51.99 %
learning rate = 0.2 , accuracy: 50.14 %
learning rate = 0.3 , accuracy: 48.72 %
```

From the above output, we can see that the model with the best performance is gradient-boosted trees using a learning rate of 0.01. This optimal model was then used on the test set.

```
accuracy of best model on predicting negative sentiment reason in test set: 53.64238410596026 %
```

Results

My first model, multinomial Naive Bayes using TF-IDF features, was found to be the best model for predicting the sentiment of tweets. This model was trained on the training set of the *sentiment_analysis* dataset. The model had an accuracy of 95.56% when tested on the test set of *sentiment_analysis*, which was the highest out of all the models I implemented. However, when I used this same model (trained on *sentiment_analysis* training set) to predict the sentiment of the *Canadian_elections_2019* dataset, it was only able to achieve an accuracy of 51%. I believe that the main reason for this large drop in accuracy is most likely due to my model being slightly overfitted on the *sentiment_analysis* dataset. From the bar plot "Distribution of Predicted and True Tweet Sentiment of Different Political Parties", it can be seen that the true sentiment of both the Conservative Party and Liberal Party seems to be mostly negative. In contrast, the true sentiment of the New Democratic Party is mostly positive. My model, however, predicted all three political parties to have more positive sentiment than negative sentiment. The true sentiment of the political parties tells us that in the public has mainly negative views on the Conservative Party and the Liberal Party, and has mainly positive views for the New Democratic Party.

My second model, gradient boosted trees using TF-IDF features with a 0.01 learning rate, was found to be the best model for predicting the reason for negative tweets in the elections dataset. When used on the test set, this model had an accuracy of 53.64%. I believe that there are two main reasons as to why my model may have failed to correctly predict some of the negative reasons of the tweets. The first reason is that some of the negative reasons are very similar to each other. For example, "scandal" and "tell lies" are very similar to each other, which means that tweets under these two reasons most likely share a lot of key words. To back up my claim, if we look at the test set, we can see that tweet 22 is classified as "scandal" and tweet 105 is classified as "tell lies", however both of these tweets have the word "disinformation" in them. These similarities amongst different target classes may have introduced unnecessary bias into the classification model. The second reason is the presence of the "Others" category in the target class (negative reasons). This category is not a single negative reason, but is actually all of the reasons not listed combined into one. Furthermore, if we look at the test set, this "others" category is by far the biggest target class out of all of the negative reasons, with a total of 365 tweets in this category. In terms of natural language processing, this can impact the performance of the classification model by introducing unnecessary variance into the data, since tweets in this class will most likely have a very wide variety of different words amongst them.

One way that I can improve my first model (predicting sentiment) is to account for the order of the words in the tweets. Having the machine learning model learn the syntactic information of the tweets will most definitely improve accuracy. A common way to do this is to treat each tweet as a sequence of individual word vectors and train it through a convolutional neural network. A way to improve my second model (predicting negative sentiment reason) would be to increase the maximum features of my TF-IDF vectorizer. Having more features means that the machine learning model will be able to train using a wider variety of words, which will most likely improve the accuracy of the model. Different classes having similar words between each other will not be as big of an issue anymore if number of features are increased, since there are more words for the model to train on.