1. Feature Construction

In order be able to use the data in the 'raw_tweets.txt' file of the Abortion Twitter Data, the data needed to first be loaded into a data frame so that we could easily interpret it. This conversion from text to data frame was handled by the Pandas library, which allows us to import files in either a CSV or txt format and then load them into what is known as a data frame. This data frame is essentially a table of values which is populated from our raw data file. The loading of the 'rawtweets.txt' file is done through the following line of code:

```
raw_tweets_df = pd.read_table('AbortionTwitterData/raw_tweets.txt',  sep='\t')
```

In order for our classifier to be able to effectively make sense of the data in the Pandas data frame, the data frame needed to undergo a couple of preprocessing steps.
The first step of preprocessing was assigning labels to the 'Tweet Text' strings in the raw_tweets_df data frame. This was done by first reading in the 'ground_truth.txt' file (which contains classification labels for each tweet in the corpus, per tweet id) and adding those labels as a new column in the raw_tweets_df data frame:

```
ground_truth_df = pd.read_table('AbortionTwitterData/ground_truth.txt',  sep='\t')
```

```
raw_tweets_df['Label'] = "Neutral"
for row in range(ground_truth_df.shape[0]):
    raw_tweets_df.loc[row, 'Label'] = ground_truth_df.loc[row, 'Label']
```

This effectively adds a 'Label' column in our main data frame, and our data frame now contains Tweet IDs, Tweet Text, and finally Labels.
Continuing with preprocessing, the second step was converting all Tweet Texts to lowercase and removing punctuation, for better classification and uniformity with regards to case recognition:

```
raw_tweets_df['Tweet Text'] = raw_tweets_df['Tweet Text'].str.lower()
raw_tweets_df['Tweet Text'] = raw_tweets_df['Tweet Text'].str.replace('[^\\w\\s]', '')
```

After this was done, the third preprocessing step was to remove stop-words, such as articles, prepositions, and pronouns. Stop-word removal helps to eliminate vocabulary noise in the data, allowing the classifier to focus on keywords and ignore extremely common words used in general English speech:

```
stop_words = set(stopwords.words("english"))
raw_tweets_df['Tweet Text'].apply(lambda x: [item for item in x if item not in stop_words])
```

In the final few steps of preprocessing, I tokenized Tweet Texts into single words through the use of the nltk library tokenizer. Tokenization is important, as it allows each of the words in the data frame to be analyzed as a separate instance. After tokenization, I had initially elected to use stemming via the nltk Porter stemmer, but this showed certain inaccuracies so I finally elected to not use stemming. By removing prefixes and suffixes to words and reducing them to their base forms, many commonly occurring words were easily classified; however, some of the most important words such as "prolife", "prochoice", and "abortion" were reduced to "life", "choice" and "abort". It can be seen from these reductions that stemming reduces the words too much, so much so that the intended meanings of the words are lost.

Finally, I used term frequency-inverse document frequency (TF-IDF) in perhaps the most important step of the preprocessing. TF-IDF essentially assigns weights to each word in the data frame, based on their occurrences and their importance to the overall corpus. The implication of this is that our model will have a much easier time distinguishing neutral, for-abortion, and against-abortion words based on their weightage in each classification pool.[1]

I elected to use the TF-IDF feature collection method for a couple of reasons. Firstly, TF-IDF has the potential to achieve higher accuracy in classifying words, due to its nature of assigning weights to words with respect to their importance in the phrases they are used in. This as opposed to, say Bag-of-Words or Length-of-Words will prove much more accurate, as neither BOW or LOW (or any other simpler word classifier) can as accurately identify the weightage or importance or commonly used words.[2] The second reason I used TF-IDF is that with larger a corpus, the simpler feature extraction methods tend to lose their effectiveness.

2. Description of the Classifier

The classifier I chose to use in this problem is the Naïve-Bayes classifier. The Naïve-Bayes classifier is built upon Bayes rule, which defines rules for conditional probability.[3] As such, this type of classifier is extremely useful in a classification problem that allows for the use of conditional probability to determine the likelihood of an occurrence. In our case, this is the likelihood of a tweet being for-abortion, against-abortion, or neutral given the occurrence(s) of certain word(s). For example, we if we want to see whether a given message is against-abortion, we can define the probability of the message being against-abortion given the words 'prolife' and 'catholic': P(against-abortion | 'prolife) * P(against-abortion | 'catholic'). The resulting probability is the likelihood estimate that the tweet is against-abortion, allowing us to classify it as so if it is above a certain probability threshold.

Naïve-Bayes works much better in this problem compared to other classifiers such as a decision tree, kNN, neural networks, or even logisitic regression. Decision trees would have too many features to split on for this problem, essentially requiring a split on each and every word. The tree would be too large and would be heavily overfit to the training data (due to so many splits and levels in the tree). The kNN and neural network could potentially be effective in forming groupings of words, but would have a difficult time classifying whether the words are spam or ham. Finally, logistic regression could in theory work for this problem, but would not be as effective as a Naïve-Bayes classifier, since the output of a logistic regression classifier is a boolean value, not a probability.[3] This could lead to many misclassified messages, since there are no likelihood probabilities considered, only hard true or false values.

3. Evaluation Technique

In order to evaluate the effectiveness of the Naïve-Bayes classifier, I took the mean of correctly predicted Tweet Text words. This essentially consisted of taking the mean of 'For Abortion', 'Against Abortion' and 'Neutral' classifications in the predictions array, and comparing them against true occurrences of such Tweet Texts.

Furthermore, to evaluate the performance of the Naïve-Bayes classifier in a more wholistic manner, I calculated the accuracy, precision, and recall values through the scikit-learn metrics library. Overall, a single run of my model returned an accuracy of around 0.5875, a precision of 0.7278, a recall of 0.5213, and an F1 score of 0.4970.

In the single run, both the accuracy and precision values are considerably lower than favorable success rates, showing that the classifier is subpar in classifying Tweet Texts, as well as inconsistent in its predictions. The high recall value shows us that our classifier is very good at learning from the training

data, being able to recall its past labels only around 52.13% of the time. Finally, the F1 score is the harmonic mean of recall and precision, essentially comparing the amount of true positives and true negatives against false positives and false negatives in classifications.[4] The F1 score of 0.4970 that the Naïve-Bayes classifier achieved for this problem shows that the classifier is moderately good at classifying spam.

As with many learners, the use of cross-validation in training helps to avoid overfitting the model to the training data. The use of k-fold validation would be useful for Naïve-Bayes in helping to avoid overfitting, a only portions of the dataset are being exposed to the classifier at a time, while the classifier is also being tested on the remaining portion of the dataset. After doing a single run of the Naïve-Bayes classifier, I also implemented a k-fold cross validation series, with 10 folds, with the hopes of improving test accuracy over the course of multiple iterations (further details on this are discussed in the following section of this report).

4. Implementation

a. Preprocess

As mentioned in the **Feature Construction** section, there were numerous preprocessing steps to prepare the data for training the classifier. The first step of preprocessing was assigning the correct classification labels for the Tweet Texts:

```
ground_truth_df = pd.read_table('AbortionTwitterData/ground_truth.txt',  sep='\t')
raw_tweets_df['Label'] = "Neutral"
for row in range(ground_truth_df.shape[0]):
    raw_tweets_df.loc[row, 'Label'] = ground_truth_df.loc[row, 'Label']
```

The second segment of preprocessing was to convert all Tweet Texts to lowercase, remove punctuation, remove stop-words, and tokenize individual words. The conversion to lowercase, along with the removal of punctuation and stop-words served to remove noise in the data. The tokenization was done so that individual words could be assigned classification labels, so that the classifier can associate occurrences of those words with the appropriate labels. In the final segments of preprocessing, I chose to not use stemming on the words, since stemming in this problem resulted in keywords being reduced too much, so much so that the intended meanings of the words are lost. However, I did use term frequency-inverse document frequency (TF-IDF) from sklearn's TfidfTransformer library. TF-IDF essentially assigns weights to each word in the dataframe, based on their occurrences and their importance to the overall corpus. The effect of this is that our model will more accurately distinguish for-abortion, against-abortion, or neutral words based on their weightage in each classification pool.[1]

b. Feature Extraction and External Library Use

With regards to feature extraction, the one of the main steps was early on in the preprocessing phase, in which classification values were assigned from 'ground_truth.txt' to the raw_tweets_df data frame. This was done through the Pandas library, which allows for the addition of columns to data frames. This label allowed for tokenization of words (after preprocessing), which was imported from the nltk.tokenizer library. The multinomial Naïve-Bayes classifier was imported from the Scikit-Learn library, along with a count vectorizer, tf-idf transformer, accuracy/f1/precision/recall scores, and also the k-fold cross validation. Finally, I imported and used Matplotlib's pyplot and wordcloud's wordcloud libraries to display and save most commonly occurring words for each of the three categories.

With regards to the memos, I qualitatively analyzed these alongside the results of the wordclouds; I basically used the memos to verify the findings in each of the wordcloud images and check whether each wordcloud contained the expected words for each category.
No normalization was used, since I was using a classifier, as opposed to a regression model. Had I been using a regression model, such as a logistical regression, then I might have needed to normalize or standardize each feature to avoid high variance.

c. Implementation of Classifier

Pandas was used to read in both the raw tweets and the ground truth text files as data frames, and then assign values from the ground truth text file to the raw tweets data frame (for later manipulation). The NLTK library was used for stop-word removal and tokenization of individual words in the Tweet Text column of the raw_tweets_df data frame.

A few different libraries from Scikit-Learn were used in preprocessing, as well as for the main classification model used in this analysis: The CountVectorizer and TfidfTransformer were used to assign weights and vectorize words in the Tweet Texts, allowing for our classifier to prioritize these first in training. The two other imports from Scikit-Learn were the multinomial Naïve-Bayes classifier and the k-fold cross-validation mechanism. The Naïve-Bayes acted as the main classifier that was trained and tested using our weighted Tweet Text words and labels, and the k-fold cross-validation was used to see if cross-validation fared better than a single test instance of the Naïve-Bayes.

Finally, I used Matplotlib's pyplot and wordcloud's wordcloud libraries to visualize the weights of commonly occurring words in each of the classification categories.

d. Cross-Validation

With regards to k-fold cross-validation, I used a 10-fold cross validation mechanism:

```
k_fold = KFold(len(y_train), n_folds=10, shuffle=True, random_state=0)
clf = model
print "k-fold cross validation score for k=10"
print cross_val_score(clf, X_train, y_train, cv=k_fold, n_jobs=1)
```

This allows for the classifier to be trained on different segments of the Tweet Text data, allowing it to be exposed to differently weighted words in different orders, along with their corresponding labels, which should help reduce overfitting.

d. Performance Metrics

The performance metrics used to evaluate the main classification model, the multinomial Naïve-Bayes, are accuracy , f1 score, precision, and recall. These were all imported as libraries from Scikit-Learn's metrics library.

With regards to the metrics from the k-fold cross-validation runs, a k-fold cross-validation score was computed using Scikit-Learn's cross_validation library.

5. Analysis of Results

After a successful single run of the classification simulation, the following output can be observed:

```
Accuracy is
0.5875
F1 Score is
0.4970355731225296
Precision Score is
0.7277676950998185
Recall Score is
0.521344537815126
```

It is apparent that the multinomial Naïve-Bayes classifier has quite a difficult time correctly classifying nearly half the tweets in the dataset. Such low accuracy, precision, and recall scores are most likely due to commonly occurring words for each of the classification categories occurring in more than one category. For example, the repeatedly occurring words ‘prolife’, ‘prochoice’, ‘abortion’, and ‘baby’ occur in all three categories. The classifier will see these words and attempt to classify them a certain way but the true classification may be some other category (see the wordclouds shown below).

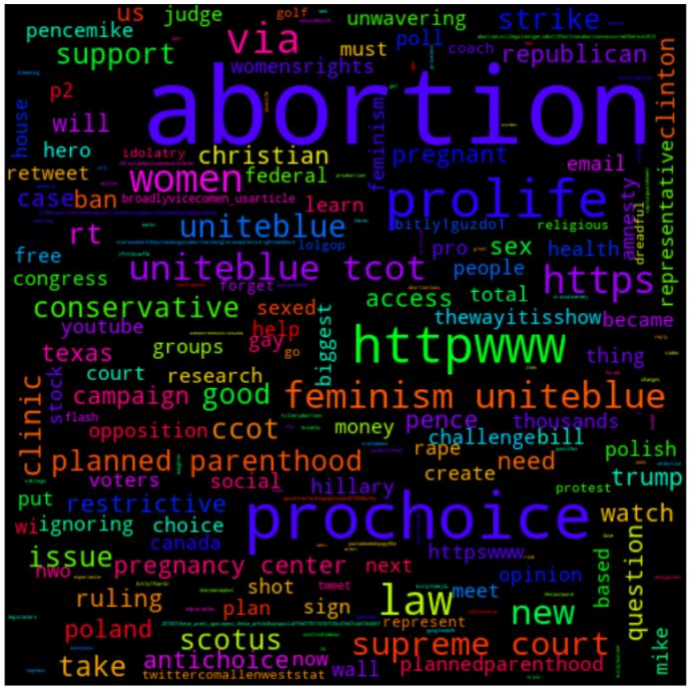

Figure 1: Neutral

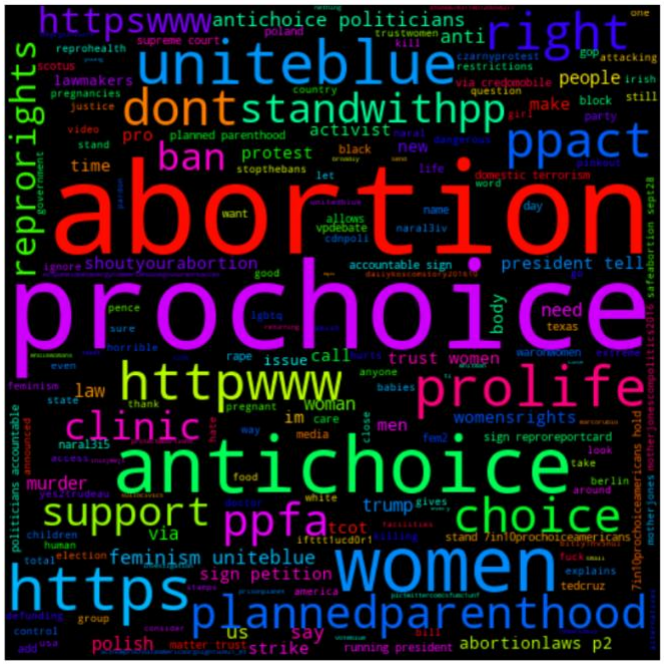

Figure 2: For-Abortion

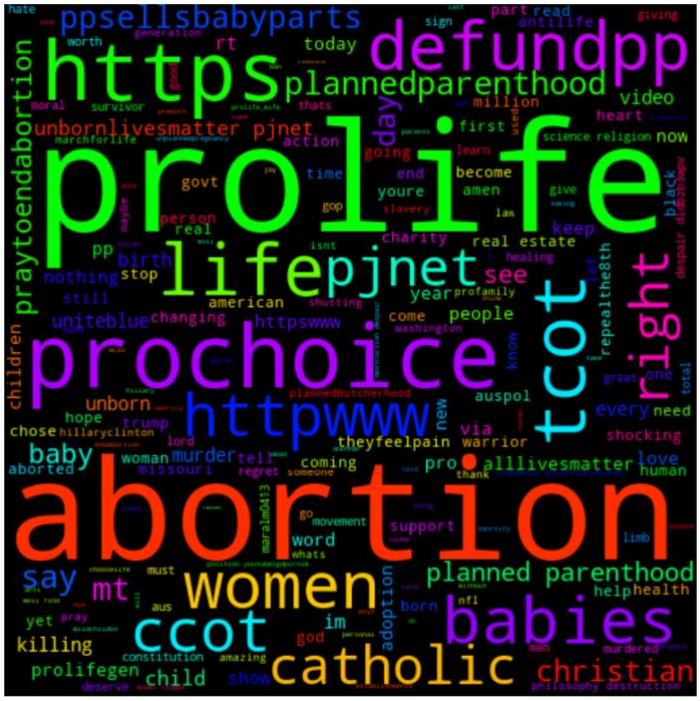

Figure 3: Against-Abortion

As the above wordclouds for the Neutral, For-Abortion, and Against-Abortion categories show, the most commonly occurring words are present in tweets from all categories. This proves a steep challenge for any classifier attempting to predict tweet sentiment through keywords, since the sharing of words between classification categories will cause a high rate of misclassified tweets.

Adding to the results of a single run of our classifier, the results of k-fold cross validation are as follows:

```
k-fold cross validation score for k=10
[0.4375  0.59375 0.53125 0.5000 0.59375 0.65625 0.53125 0.59375 0.65625
 0.59375]
```

Even with k-fold cross validation, it the low values and randomness in test accuracy shows that the model has difficulty in correctly identifying certain tweets. It can be inferred that the low accuracy is not a result of overfitting, but rather a difficult classification problem. Once again, this may be due to keywords occurring heavily in all three classification categories as is apparent in the above wordclouds.

6. Replication and Comparison

As shown above, a single run of my classifier returned an accuracy of about 58.75%, and the k-fold cross-validation returned an average of 56.875% and best accuracy of 65.625%. For both mean accuracy and best accuracy, this classifier performed worse than the one described in section 4.1 of the original study.[5] However, it is important to note that even the classifier in the original study had rather low accuracy numbers, with a mean of 67% and best of 81%. When compared to, say the ham/spam text classification problem from the previous assignment (which returned a mean accuracy of ~97%), this problem seems to be much more difficult to effectively classify. When compared a single run of my classifier (Naïve-Bayes), accuracy (0.5875), f1(0.49), and recall(0.52) are all scoring less than those for the SVM in the original study.[5] Interestingly, the precision for my Naïve-Bayes is much better than the one for the SVM in the study (0.7278 vs. 0.67). However, this simply means that on average, the Naïve-Bayes tends to make very similar classifications and does not necessarily imply accuracy.

With regards to the provided memos, the memos were, in my case, simply used to qualitatively compare against the wordclouds formed, and therefore do not result in any statistical improvements over the figures from the original study. When comparing the hashtags in these memos, numerous vocabulary instances can also be seen in the respective wordcloud. For example, in the Against-Abortion wordcloud, words such as ‘shameful’, ‘unborn’, and ‘defundpp’ are present, which are also present in memo4.txt. The topical hashtags in memo3.txt correlate with the three most commonly occurring words in all wordclouds, ‘abortion’, ‘prolife’, and ‘prochoice’, with the only missing term in memo3.txt being ‘babies’.

In conclusion, it is interesting to observe tweets, or any online social activity for that matter, on the same issue and with the same buzzwords present. As has been observed in this study as well as the original study by De Choudhury et al., building a classifier around such issues can prove to be a challenge, as opposing sides of an argument or issue could use the same terminology and same patterns of communication. Perhaps we can learn from this, and in a way realize that people on strongly differing sides of an issue have more in common than is apparent.

Works Cited

1. TF-IDF Documentation. http://www.tfidf.com
2. Sycorax. TF-IDF Explanation. https://stats.stackexchange.com/questions/153069/bag-of-words-for-text-classification-why-not-just-use-word-frequencies-instead
3. Deb, S. “Naïve-Bayes vs Logistic Regression.”https://medium.com/@sangha_deb/naive-bayes-vs-logistic-regression-a319b07a5d4c
4. Wikipedia. “F1 Score.” https://en.wikipedia.org/wiki/F1_score
5. Sharma, E., Saha, K., Ernala, S. K., Ghoshal, S., & De Choudhury, M. (2017). Analyzing Ideological Discourse on Social Media: A Case Study of the Abortion Debate. In Proceedings of CSSA's Annual Conference on Computational Social Science (CSS) 2017.