**US Airline Sentiment Analysis**

|  |  |
| --- | --- |
| **Author 1: Ann Nhu Nguyen**  **Author 2: Hiren Rathod**  **Author 3: Sandra Lin** |  |

**1** **Introduction/Background (Hiren)**

In an effort to expand our understanding of text classification, sentiment analysis, and textual feature representation, our group has conducted research on various text classification strategies to classify a corpus of United States airline tweets. We have compared the baseline naive bayes classification strategy with the collocation classification method along with multiple term frequency - inverse document frequency weighted approaches to sentiment classification. This project would be useful for students studying text classification and this project can benefit those who are interested in implementing differing machine learning algorithms to improve text classification accuracy. This project has definitely been performed before. Our group has decided to broaden the scope of our naive bayes sentiment classification assignment to include three different sentiments with additional sentiment classification techniques that will be discussed throughout the report. In order to understand the remaining components of this report, one must be familiar with python, tokenization, textual representations, probabilities, logarithms, the naive bayes classification algorithm, textual collocations, simple accuracy evaluation, add-one smoothing, term frequencies, and inverse document frequencies.

**2** **Corpus/Data (Ann)**

We used the Twitter US Airline Sentiment Dataset which contained tweets about US airlines and whether the sentiment of the tweets was positive, neutral, or negative. The data that was provided to us originally came from Crowdflower’s Data for Everyone library but the website we obtained the data from is a slightly modified version of the original source. It includes both a CSV file and SQLite database. There were 5000 files for the training set, 5000 files for the development set, and 4641 files for the test set.

**3** **Tasks Performed (Ann)**

Since our dataset already came with its appropriate labels, Hiren parsed the CSV file into the development set, training set, and test set. Additionally, since we were only interested in comparing the accuracy of our three different Naive Bayes classifiers, Hiren parsed only the tweets with its respective polarity. Thus, ignoring the other categories such as the name of the airline, tweet location, username, etc. Ann trained the Naive Bayes classifier which provided the blueprint to classifying the three models. In addition, Ann provided the Naive Bayes classification model which serves as the baseline that we will compare the other two classification models with in terms of its accuracy. Hiren provided the second classification model that included tf-idf weights. Sandra worked on the last classification model that specifically dealt with the use of collocations. Once we had all of our classification models ready, we ran our classifiers to an output file. From there, Hiren was able to compute the accuracy for all 3 classifiers.

**3.1 Naive Bayes Classifier - Baseline Model**

Text classification has many useful applications. Our project specifically focuses on one of them - sentiment analysis. Sentiment analysis allows us to gain an overview of public opinion, whether positive or negative, regarding any topic. Understanding feedback from those using a particular service is critical in improving the experience and satisfaction of customers. A way to do sentiment analysis is by implementing a classifier, specifically the Naive Bayes classifier. We chose this specific classifier because it’s common and well understood. Importantly, it’s a great classifier to use when comparing it to other classifiers which is slightly what we’re doing, but instead we’re comparing the same type of classifier but incorporating different feature representations and improvements. With the standard Naive Bayes classifier, we’re using a specific set of features called the bag-of-words which is how our classifier will represent the text. The bag-of-words is all of the tokens in the document with the counts but doesn’t account for word order. So the classifier will see all of the tokens and the number of times each token appears but is clueless on the order they were in originally. Since the classifier is a probabilistic model that is based on Bayes’ theorem, our classifier will find the probability that a specific tweet regarding major US airlines is of positive, neutral, or negative sentiment given the words. The way the Naive Bayes classifier does this is by building a model of what positive, neutral, and negative reviews look like. Therefore, before building the baseline classifier, a training set was built which consisted of several training examples with tweets about major US airlines with its appropriate label. Next, we have the development set which is a sample of tweets that are similar in the training set and test set. The development set is what we will be running our model on to give us an idea of whether our model is working well or not. Lastly, we have our test set which evaluates the model we trained by using the reviews we didn’t use to train our model. After performing these steps, the baseline Naive Bayes classifier was constructed and consisted of the frequency distributions for each class (positive, neutral, and negative) as well as the number of samples in each class. In addition, the classifier computed the prior probabilities and conditional probabilities for each of the words. The classifier also included a smoothing technique also known as add-one smoothing which prevents zero probability in Naive Bayes. Without smoothing, classification could be poor.

One disadvantage of the Naive Bayes classifier is that it has the assumption that the features in a dataset are mutually independent. We use this assumption although it doesn’t coincide with our knowledge of language and limits the applicability of this algorithm in real-word cases. Although this independence assumption allows the classifier to work in a simple way, the approximations that the classifier takes in may end up hurting our classification. As you will see, we implement 2 additional Naive Bayes classifiers that contain improvements that could potentially increase the accuracy of the classifiers.

**3.2** **Tf-idf Modification Classifier**

The second classifier is similar to the baseline classifier except with the addition of tf-idf weights. Tf-idf stands for term frequency - inverse document frequency. Tf-idf is a statistical measure that evaluates how important a word is to the corpus of tweets. The meaning of the text is not just the meaning of the individual words, but also how the words are structured. The bag-of-words feature captures some of the meaning but not all. Thus, giving the features a lower or higher weight could potentially improve classification. The first term, term frequency (tf) is the number of times a word appears in a document, divided by the total number of words in that document. The second term, inverse document frequency, is computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears. Those two components would then be multiplied by each other to give us the weights.

In implementing the classifier, the uniform weight of 1, that is done in add-one smoothing, will need to be replaced with a generated tf-idf weight. Out of the many tf-idf permutations, the term frequency (tf) formulas explored in this project are: raw count term frequencies, raw count and length based term frequencies, boolean term frequencies, augmented term frequencies, and logarithmically scaled term frequencies. The raw count term frequency is how many times a type appears in a corpus. The length based term frequency is the average between the raw count term frequency and the number of tokens in the file. The boolean term frequency returns the value 1 if the token is present in the file and 0 otherwise. The logarithmically scaled term frequency scales the raw count term frequency logarithmically. The augmented term frequency is the most involved term frequency formula to evaluate. The intuition here is to take away the effect of files that are much longer than others (Wikipedia).

In order to represent the values needed to compute the tf-idf weights, the model data structure needs to incorporate additional components in the training phase. It is necessary to create 3 empty dictionaries that contain mappings from tokens to counts. Each dictionary is created to facilitate different mappings between tokens and file counts in the positive, neutral, and negative training sets. This mapping will ultimately represent how many files a particular token occurs in. When computing the inverse document frequency, the token can be used as the key to retrieve the number of documents where the token appears. As the code traverses each file within the positive files training set, a temporary dictionary is created to hold tokens that have already been seen. This is done so that recurring tokens do not increase their count value within the same file. If the token is seen for the first time within the mapping between tokens and file counts and the token is also seen for the first time in the temporary map holding visited tokens, then the token is appended to the temporary map and the file count dictionary with a file count of 1. If the token exists in the mapping between tokens and file counts but it does not exist in the temporary map, then we increment the tokens’ file count value and we add it to the temporary map of visited tokens. If the token exists in the mapping between tokens and file counts and it does exist in the temporary map of visited tokens, then we move to the next token in the list of file tokens. Dictionaries are used here instead of a list to improve O(n) searching within a list to a O(1) key-value lookup in the dictionary. These steps are repeated with their corresponding file count dictionaries as we also traverse through the neutral and negative files training sets.

**3.3** **Collocation Modification Classifier**

The last classifier is also the same as the baseline classifier but with the use of collocations. Collocation is the expression of multiple words that frequently occur together in a corpus.They are word combinations that occur together more often than by chance. This classifier was able to find the 100 most common collocations within the training set and test set. This was done by finding the unigrams (single word) frequency distribution of every word that is found in the training set and test set. Additionally, finding the bigram (adjacent words) frequency distribution of every two words that is found in the training set and test set. From looking at every adjacent word combination created in the bigram frequency distribution, a score was calculated in respect to how often those two words were used side by side. The formula used to find the score of the collocation is P(w1w2) / (P(w1) \* P(w2)) which is the probability of two words co-occurring divided by the probability of one of the words occurring by itself and multiplied by the probability of the other word occurring by itself. This formula also only looked at individual words that have only appeared more than 100 times in the whole corpus including the training set and test set. Next, the list of of collocations generated by the classifier was sorted from lowest score to highest score, taking only the 100 highest scoring collocations. Then a new training set was made with the 100 most common collocations. Also, we created a multi-word expression token using MWETokenizer that was imported from nltk.tokenize. From this, we were able to tokenize each text file in the positive, neutral, and negative training set. We found that the frequency distribution of all positive, neutral and negative files have changed while comparing to the frequency distribution found when training the text files not using collocation. Using the new training set, we were able to generate labels for all test files in the file to find the accuracy of the modified collocation classifier.

**4** **Analysis (Ann)**

From computing the accuracy of all 3 classifiers, we found that the tf-idf modification classifier had a higher accuracy than the baseline classifier with an increase of at most 1.2%. This was not the case for the collocations modification classifier as it did worse than the baseline classifier with a difference of 2%. The baseline classifier had an accuracy of 79.078%. With the tf-idf modification classifier, we computed 5 different accuracies due to implementing different term frequency (tf) formulas and all of them surpassed the accuracy for the baseline classifier. The raw count term frequency had an accuracy of 80.026%. The raw count and length average term frequency had an accuracy of 79.552%. The boolean term frequency had an accuracy of 80.047%. The augmented term frequency had an accuracy of 80.263%. Lastly, with the highest accuracy for the tf-idf modification classifier is the logarithmically scaled term frequency with an accuracy of 80.306%. We believe that this is the highest accuracy because we are already calculating logarithms within the classifier. Next, we have the collocation modification classifier with an accuracy of 77% which is lower than the baseline. These results ultimately show that the Naive Bayes classifier can do surprisingly well although fails to demonstrate how we actually use language in real life due to the strong independence assumption. With that said, improvements can be made, however, only by a small percentage. In respect to the collocation modification classifier, we are still unsure as to why it performed worse than the baseline classifier. We had hoped that all of these improvements that were made would increase the accuracy. Given more time, we could research if others had problems with low accuracy when implementing collocations in their Naive Bayes classification. Furthermore, we understand that not all strategies are guaranteed to improve accuracy. Consequently, from the results, it is clear as to why people choose this type of classifier as it is simple yet efficient.

**5** **Conclusion (Sandra)**

With the little improvement that was seen using the tf-idf model and the accuracy for the collocation modified classifier being lower than the baseline accuracy, we are able to conclude that not all modifications to the baseline classifier would necessarily improve the accuracy. Even when there is an improvement, it could be almost just the same as the baseline Naive Bayes classifier. In some cases, the baseline classifier could perform better than other models. While our modifications did not improve significantly on the accuracy of the classifier, we are not able to conclude that the baseline Naive Bayes classifier would be the best model to use for our corpus. The Naive Bayes classifier uses independent assumptions which would be beneficial if we were studying a corpus with independent features.

Due to the time constraints, we were only able to work with the two modifications listed above. Given more time we would have liked to look into combining multiple modifications including removing stop words, adding tf-idf weights, collocations, etc… Since we were able to make some improvements with our separate modifications we think it would be possible to improve the accuracy even more while combining different methods. Another method we would be interested in using to improve the accuracy of our classifier is by using the perceptron. With the perceptron, since there is not an independent assumption for the feature in the corpus, it can give us a better understanding of the relationship between the features. The perceptron is known to be a discriminative classifier while Naive Bayes is considered a generative classifier. The idea is that with the perceptron classifier, we would be able to give each feature a feature weight. By adding all the frequency distribution of the tokens multiplied by the feature weight, it would give us a score for the feature. With the score produced, the perception will then try to classify each example one at a time, and as soon as it detects a mistake, it would change the weight. With this method, we would be looking at the dependence of the features rather than the independent frequency distribution of the features, which would hopefully give us a better accuracy compared to the Naive Bayes classifier.

**6** **Ethical Considerations (Sandra)**

The information that was gathered and studied would mostly benefit US airline companies as it gives them a perspective from the customer's point of view. With the results we produced, it could potentially give US airline companies a better understanding on what they need to work on in order to improve customer satisfaction. We do not believe the results of this study would cause any potential moral or ethical issues since all data was captured through social media, Twitter, and is considered public information.

The amount of negative comments we end up obtaining was more than double the amount of positive and neutral comments combined. With that being said, the bias that was observed from the data we used throughout this research is that people tend to leave negative comments over positive comments. It would most likely take an unique or extraordinary experience for customers to leave a positive and or satisfactory comment. Therefore the data we had was biased to begin with, causing us to have less training data for positive and neutral parts of the research.

**Reference**

“Tf–Idf.” *Wikipedia*, Wikimedia Foundation, 8 Mar. 2021, en.wikipedia.org/wiki/Tf%E2%80%93idf.