JSC270 Winter 2023 Assignment 4
Natural Language Processing (NLP)

Group 12: Huayin Luo & Mojan Majid

Colab:
https://colab.research.google.com/drive/1mEj68G-ltnWPW-epSvXWHDf2PjT6hXlN#scrollTo=1cP3Hh0JM4Pk

Breakdown of work:
Worked on part 1 together, and part 2 building the model together. As Mojan was sick for the later half, Huayin worked on the presentation and finalizing the report.

Part I: Sentiment Analysis with a Twitter Dataset (20 pts)

A) (1 pt) Consider the training data. What is the balance between the three classes? In other words, what proportion of the observations (in the training set) belong to each class?

We map Extremely Negative and Negative sentiments to 0, Neutral to 1, and Positive and Extremely Positive to 2.

There are 15398 observations with sentiment 0, 7713 with sentiment 1, and 18046 with sentiment 2. Thus, the proportions for sentiments 0, 1, 2 in order are 0.37, 0.19, and 0.44.

B) (1 pt) Tokenize the tweets. In other words, for each observation, convert the tweet from a single string of running text into a list of individual tokens (possibly with punctuation), splitting on whitespace. The result should be that each observation (tweet) is a list of individual tokens.

See function *tokenize* in the colab file.

C) (1 pt) Using a regular expression, remove any URL tokens from each of the observations.

Hint: In this dataset, all such tokens begin with “http”.

See function *remove_url* in the colab file.

D) (2 pts) Remove all punctuation (,.?!;:’") and special characters(@, #, +, &, =, $, etc). Also, convert all tokens to lowercase only. Can you think of a scenario when you might want to keep some forms of punctuation?

See function *remove_punctuation* and *convert_lowercase* in the colab file.

E) (1pt) Now stem your tokens. This will have the effect of converting similar word forms into identical tokens (e.g. run, runs, running → run). Please specify which stemmer you use. Note: There are several different stemmers available through nltk and Scikit-learn. I recommend the Porter stemmer, but you may use a different one if you wish.

See function *stem_tokens* in the colab file.

F) (1pt) Lastly, remove stopwords. Using the english stopwords list from nltk, remove these common words from your observations. This list is very long (I think almost 200 words), so remove only the first 100 stopwords in the list.

See function *remove_stopwords* in the colab file.

G) (2 pts) Now convert your lists of words into vectors of word counts. You may find Scikit-learn's CountVectorizer useful here. What is the length of your vocabulary? Hint: The matrix of counts will be D × V , where D is the number of documents (tweets), and V is the number of features (word counts).

The length of our vocabulary is 1000 (the actual length of the vocabulary is higher but we have used max_features = 1000 in this question).

H) (4 pts) Recall the definition of the Naive Bayes model. If each document (tweet) is a collection of words (w1, · · · , wN) belonging to class Ck (k = 0, 1, 2), then the Naive Bayes approach models the probability of each tweet belonging to class k:

$$P(C_k|w_1, \cdots, w_N) \propto P(w_1, \cdots, w_N|C_k)P(C_k)$$
$$= P(C_k)\Pi_{i=1}^{N} P(w_i|C_k)$$

The last equality follows from our "naive" assumption that words are conditionally independent given class. The probabilities are estimated using the frequencies of words within each class (bag of words), and we assign the class label according to which of the 3 posterior class probabilities (P(Ck|w1,⋯ ,wN)) is the highest.

Fit a Naive Bayes model to your data. Report the training and test error of the model. Use accuracy as the error metric. Also, report the 5 most probable words in each class, along with their counts. You might find Scikit-learn's MultinomialNB() transformer useful. Use Laplace smoothing to prevent probabilities of zero.

We fit the Naive Bayes model with our training dataset. Then, we tested the accuracy of the model on our test/training datasets. The accuracy of the model on the training dataset was 67.82%, and the accuracy of the model on the test dataset was 38.39%.

The 5 most probable words in each class and their counts were as follows:

Class 0: coronaviru (6703), covid19 (4862), price (4332), food (3623), thi (3206)

Class 1: coronaviru (3792), covid19 (2752), store (1581), supermarket (1436), price (1361)

Class 2: coronaviru (7467), covid19 (6003), store (3896), food (3772), thi (3323)

I) (2 pts) Would it be appropriate to fit an ROC curve in this scenario? If yes, explain why. If no, explain why not.

A ROC curve shows the performance of a binary classifier so it is not appropriate in this scenario where we have 3 classes.

J) (2 pts) Redo parts G-H using TF-IDF vectors instead of count vectors. You might find Scikitlearn's TfidfVectorizer() transformer useful. Report the training and test accuracy. How does this compare to the accuracy using count vectors?

The accuracy of the model on the training dataset was 66.39%, and the accuracy of the model on the test dataset was 40.71%.

In comparison with the results using count vectors, the accuracy of the model was slightly better for the testing dataset, and slightly worse for the training dataset.

K) (3 pts) Recall lemmatization converts each word to its base form, which is a bit stronger than simply taking the stem. Redo parts E-H using TF-IDF vectors instead of count vectors. This time use lemmatization instead of stemming. Report train and test accuracy. How does the accuracy with lemmatization compare to the accuracy with stemming? Note: Like stemmers, there are multiple lemmatizers you might use. We recommend the WordNet lemmatizer offered by nltk.

The accuracy of the model on the training dataset was 66.40%, and the accuracy of the model on the test dataset was 43.18%.

Using lemmatization, in comparison with the results using stemming, the accuracy of the model was better for the testing dataset by almost 3%, and slightly better for the training dataset.

Bonus **(1 pt): Is the Naive Bayes model generative or discriminative? Explain your response.**

Naive Bayes is a generative model.

As we know, generative models and discriminative models can both serve as classification models. However, while discriminative models determine a divisive barrier

based on the data, generative models can model the actual data distribution based on the data.

That is, generative models do not determine the conditional probability P(Y|X) directly from the data. Instead, they use P(Y) and the joint probability P(X, Y) to calculate the conditional probability using Bayes' rule. As we can see, this is exactly the type of approach the Naive Bayes model takes.