# Twitter Sentiment Analysis

## Project Description

The goal of this project is to analyze online data to measure customer satisfaction for the airlines in the US. To achieve this, we will be using simple yet effective algorithm i.e. Naive Bayes because it has proven to work well in the field of Natural Language Processing (NLP). We will compare the results of different version of Naive Bayes like Multinomial vs Bernoulli Naive Bayes. Also, we will tweak the algorithm to achieve even better accuracies using Dirichlet Smoothing.

## Dataset

The dataset used is a preprocessed and modified subset of the Twitter US Airline Data Set available on kaggle [1]. It is based on over 14000 real tweets about airlines in US. Tweets have been preprocessed in the following ways:

• Stop word removal: Words like “and”,“the”, and “of”, are very common in all English sentences and are therefore not very predictive. These words have been removed from the tweets.
• Removal of non-words: Numbers and punctuation have both been removed. All white spaces (tabs, newlines, spaces) have all been trimmed to a single space character
• Removal of infrequent words: Words that occur only once in all data set are removed from tweets in order to reduce the size of the data.

The data has been already split into two subsets: a 11712-tweet subset for training and a 2928-tweet subset for testing The features have been generated for you. You will use the following files:
• question-4-train-features.csv
• question-4-train-labels.csv
• question-4-test-features.csv
• question-4-test-labels.csv
• question-4-vocab.txt

The files that ends with features.csv contains the features and the files ending with labels.csv contains the ground truth labels. In the feature files each row contains the feature vector for a tweet. The j-th term in a row i is the occurrence information of the j-th vocabulary word in the i-th tweet. The size of the vocabulary is 5722. The label files include the ground truth label for the corresponding tweet, the order of the tweets (rows) are the same as the features file. That is the i-th row in the files corresponds to the same tweet. Any tweet is labeled as either positive, negative or neutral. The file ending with vocab.txt is the vocabulary file in which the first element in j-th is the word that j-th feature represents and the second element is the term

frequency of the word in the all data set (training and test sets) regardless of the classes (positive, neutral and negative).

**Bag-of-Words Representation and Multinomial Naive Bayes Model**

The bag-of-words document representation makes the assumption that the probability that a word appears in tweet is conditionally independent of the word position given the class of the tweet. If we have a particular tweet document Di with ni words in it, we can compute the probability that Di comes from the class yk as:

$$\mathbf{P}(D_i \,|\, Y = y_k) = \mathbf{P}(X_1 = x_1, X_2 = x_2, .., X_{n_i} = x_{n_i} \,|\, Y = y_k) = \prod_{j=1}^{n_i} \mathbf{P}(X_j = x_j \,|\, Y = y_k) \qquad (4.1)$$

In Eq. (4.1), Xj represents the j th position in tweet Di and xj represents the actual word that appears in the j th position in the tweet, whereas ni represents the number of positions in the tweet. As a concrete example, we might have the first tweet (D1) which contains 200 words (n1 = 200). The document might be of positive tweet (yk = 1) and the 15th position in the tweet might have the word “well” (xj = “well”). In the above formulation, the feature vector X~ has a length that depends on the number of words in the tweet ni . That means that the feature vector for each tweet will be of different sizes. Also, the above formal definition of a feature vector ~x for a tweet says that xj = k if the j-th word in this tweet is the k-th word in the dictionary. This does not exactly match our feature files, where the j-th term in a row i is the number of occurrences of the j-th dictionary word in that tweet i. We can slightly change the representation, which makes it easier to implement:

$$\mathbf{P}(D_i \,|\, Y = y_k) = \prod_{j=1}^{V} \mathbf{P}(X_j \,|\, Y = y_k)^{t_{w_j,i}} \qquad (4.2)$$

where V is the size of the vocabulary, Xj represents the appearing of the j-th vocabulary word and twj ,i denotes how many times word wj appears in tweet Di . As a concrete example, we might have a vocabulary of size of 1309, V = 1309. The first tweet (D1) might be a positive tweet (yk = 1) and the 80th word in the vocabulary, w80, is “amazing” and tw80,1 = 2, which says the word “amazing” appears 2 times in tweet D1. Notice that these two models (Eq. (4.1) and Eq. (4.2)) are equivalent.

In the classification problem, we are interested in the probability distribution over the tweet classes (in this case positive, negative and neutral tweets) given a particular tweet Di . We can use Bayes Rule to write:

$$\mathbf{P}(Y = y_k | D_i) = \frac{\mathbf{P}(Y = y_k) \prod_{j=1}^{V} \mathbf{P}(X_j \,|\, Y = y)^{t_{w_j,i}}}{\sum_k \mathbf{P}(Y = y_k) \prod_{j=1}^{V} \mathbf{P}(X_j \,|\, Y = y_k)^{t_{w_j,i}}} \qquad (4.3)$$

Note that, for the purposes of classification, we can actually ignore the denominator here because we need to compare the posterior probabilities obtained for each class. Since the denominator is a constant number, dividing this number for each class will not change the ratio of posterior probabilities. Hence, we can ignore the denominator as we will classify just as same without the denominator. So, we can continue to write equation 4.3 as:

$$\mathbf{P}\left(Y = y_k | D_i\right) \propto \mathbf{P}\left(Y = y_k\right) \prod_{j=1}^{V} \mathbf{P}\left(X_j \,|\, Y = y\right)^{t_{w_j,i}} \tag{4.4}$$

$$\hat{y}_i = \arg\max_{y_k} \mathbf{P}\left(Y = y_k \,|\, D_i\right) = \arg\max_{y_k} \mathbf{P}\left(Y = y_k\right) \prod_{j=1}^{V} \mathbf{P}\left(X_j \,|\, Y = y_k\right)^{t_{w_j,i}} \tag{4.5}$$

Probabilities are floating point numbers between 0 and 1, so when you are programming it is usually not a good idea to use actual probability values as this might cause numerical underflow issues. As the logarithm is a strictly monotonic function on [0,1] and all of the inputs are probabilities that must lie in [0,1], it does not have an affect on which of the classes achieves a maximum. Taking the logarithm gives us:

$$\hat{y}_i = \arg\max_{y} \left( \log \mathbf{P}\left(Y = y_k\right) + \sum_{j=1}^{V} t_{w_j,i} * \log \mathbf{P}\left(X_j \,|\, Y = y_k\right) \right) \tag{4.6}$$

where ˆyi is the predicted label for the i-th example.

Now, having derived the theory for coding Multinomial Naive Bayes, we actually load the dataset into our IDE (SPYDER) using Python and analyse the dataset (Please refer to appendix for corresponding codes). In our training set, we have 7091 negative tweets out of total of 11712 tweets which means we have 60.5 % negative tweets. This is indeed a skewed dataset as 1 class of tweets occupies more than half of dataset. Only 40 % data is represented by other 2 classes. Training a classifier on such kind of dataset would not result in a good or efficient model i.e. it will not be able to identify the proper class most of the time as compared to the model trained on a balanced dataset. We will end up with a biased classifier towards the majority label of the training set.

Other effects include inefficient training as we will be spending time to train our classifier for unintersing features. We can possible emphasize on useless features and miss the good ones. This problem can be handled by preprocessing the data using resampling techniques. We can perform “random undersampling” of the majority class i.e. we can randomly eliminate some instance of majority class until dataset balances out. Another possible option would be the reverse of it i.e. “random oversampling ” where we can replicate instances of minority class to balance the dataset. However, as one applies these methods, one must realise that these methods might have their own implications. There exists improved versions of resampling methods.

The parameters to learn and their MLE estimators are as follows:

$$\theta_{j \mid y=neutral} \equiv \frac{T_{j,y=neutral}}{\sum_{j=1}^{V} T_{j,y=neutral}}$$

$$\theta_{j \mid y=positive} \equiv \frac{T_{j,y=positive}}{\sum_{j=1}^{V} T_{j,y=positive}}$$

$$\theta_{j \mid y=negative} \equiv \frac{T_{j,y=negative}}{\sum_{j=1}^{V} T_{j,y=negative}}$$

$$\pi_{y=positive} \equiv \mathbf{P}\left(Y = positive\right) = \frac{N_{positive}}{N}$$

• Tj,neutral is the number of occurrences of the word j in neutral tweets in the training set including the multiple occurrences of the word in a single tweet.

• Tj,positive is the number of occurrences of the word j in positive tweets in the training set including the multiple occurrences of the word in a single tweet.

• Tj,negative is the number of occurrences of the word j in negative tweets in the training set including the multiple occurrences of the word in a single tweet.

• Npositive is the number of positive tweets in the training set.

• N is the total number of tweets in the training set.

• πy=positive estimates the probability that any particular tweet will be positive.

• θj | y=neutral estimates the probability that a particular word in a neutral tweet will be the j-th word of the vocabulary, P (Xj | Y = neutral)

• θj | y=positive estimates the probability that a particular word in a positive tweet will be the j-th word of the vocabulary, P (Xj | Y = positive)

• θj | y=negative estimates the probability that a particular word in a negative tweet will be the j-th word of the vocabulary, P (Xj | Y = negative)

## Multinomial Results

Now we have everything we need to train a Naive Bayes classifier using all of the data in the training set ( question-4-train-features.csv and question-4-train-labels.csv). Refer to Appendix for python implementation. we test your classifier on the test data (question-4-test-features.txt and question-4-test-labels.txt), and report the testing accuracy as well as how many wrong predictions we made. In estimating the model parameters we use the above MLE estimator. If it arises in our code, we define $0 * \log 0 = 0$ (note that $a * \log 0$ is as it is, that is -inf ). In case of ties, we predict “neutral”.

**Accuracy = 62.81%** , **No. of wrong predictions = 1089 out of 2928**.

One can argue that MLE is not a good estimate in this situation as it exactly works on the provided training data. In case our training data is not a good representation of real underlying population distribution (skewed in our case), it might result in a bad model which kind of overfits the training data provided to the classifier.

**Dirichlet Smoothing**

To improve the accuracy and remove potential overfitting issues in our classifier, we again compute the MAP estimate of θ parameters using a fair Dirichlet prior. This corresponds to additive smoothing. The prior is fair in the sense that it "hallucinates" that each word appears additionally α times in the train set. The MAP estimates are changed and found to be:

$$\theta_{j\,|\,y=neutral} \equiv \frac{T_{j,y=neutral}+\alpha}{\sum_{j=1}^{V} T_{j,y=neutral}+\alpha*V}$$

$$\theta_{j\,|\,y=positive} \equiv \frac{T_{j,y=positive}+\alpha}{\sum_{j=1}^{V} T_{j,y=positive}+\alpha*V}$$

$$\theta_{j\,|\,y=negative} \equiv \frac{T_{j,y=negative}+\alpha}{\sum_{j=1}^{V} T_{j,y=negative}+\alpha*V}$$

$$\pi_{y=positive} \equiv \mathbf{P}\left(Y = positive\right) = \frac{N_{positive}}{N}$$

We set α = 1 and then train our classifier using all of the training set and have it classify all of the test set.we find the new results as:

**Accuracy = 75.31%, No. of wrong predictions = 723 out of 2928.**

As it is clearly visible, this additive smoothing addresses the problem in multinomial naive n=bayes and handles the overfitting issue.

## Bag-of-Words Representation and Bernoulli Naive Bayes Model

Now, we train a Bernoulli Naive Bayes classifier using all of the data in the training set ( question-4-train-features.csv and question-4-train-labels.csv). We will test our classifier on the test data (question-4-test-features.txt and question-4-test-labels.txt, and report the testing accuracy as well as how many wrong predictions were made.

Remember that this time, if the j-th word exist in i-th tweet than the related term is set to 1, and 0 otherwise, that is, tj = 1 when the word exists and tj = 0 otherwise. The formula for the estimated class is given in Eq. (4.7). In estimating the model parameters use the below MLE estimator equations.

$$\hat{y}_i = \arg\max_y \left( \log \mathbf{P}(Y = y_k) + \log(\prod_{j=1}^{V} t_j * \mathbf{P}(X_j \mid Y = y_k) + (1 - t_j) * (1 - \mathbf{P}(X_j \mid Y = y_k))) \right) \quad (4.7)$$

where, ˆyi is the predicted label for the i-th example and tj indicates whether word j appears in the document.

The parameters to learn and their MLE estimators are as follows:

$$\theta_{j \mid y=neutral} \equiv \frac{S_{j,y=neutral}}{N_{neutral}}$$

$$\theta_{j \mid y=positive} \equiv \frac{S_{j,y=positive}}{N_{positive}}$$

$$\theta_{j \mid y=negative} \equiv \frac{S_{j,y=negative}}{N_{negative}}$$

$$\pi_{y=positive} \equiv \mathbf{P}(Y = positive) = \frac{N_{positive}}{N}$$

• Sj,neutral is the number of occurrences of the word j in neutral tweets in the training set NOT including the multiple occurrences of the word in a single tweet.

• Sj,positive is the number of occurrences of the word j in positive tweets in the training set NOT including the multiple occurrences of the word in a single tweet.

• Sj,negative is the number of occurrences of the word j in negative tweets in the training set NOT including the multiple occurrences of the word in a single tweet.

• Npositive is the number of positive tweets in the training set.

• N is the total number of tweets in the training set.

• πy=positive estimates the probability that any particular tweet will be positive.

• θj | y=neutral estimates the fraction of the neutral tweets with j-th word of the vocabulary, P (Xj | Y = neutral)

• θj | y=positive estimates the fraction of the positive tweets with the j-th word of the vocabulary, P (Xj | Y = positive)

• θj | y=negative estimates the fraction of the negative tweets with the j-th word of the vocabulary, P (Xj | Y = negative)

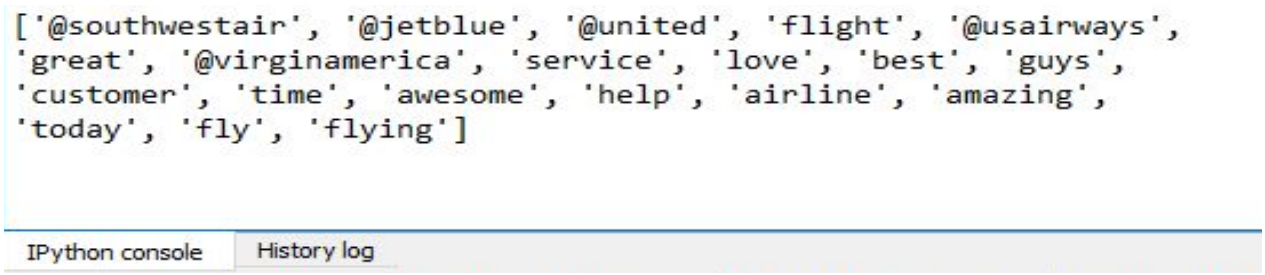
The results for Bernoulli Naive Bayes are found to be:

**Accuracy = 64.14% , No. of wrong predictions = 1050 out of 2928.**

Comparison of this (Bernoulli model) with other models shows us that the multinomial model with Dirichlet smoothing outperforms other models.

## Vocabulary Analysis

Using question-4-vocab.txt file, we now find the most commonly used 20 words in each tweet class in the training set and make comments on them. To see the whole list by class, please run the script and see the results. The top 20 common words by positive tweet class are shown in the console screenshot below:




The most common 20 words by class are, in fact, as expected and are interpretable (run and see the results of codes). For example, most common 20 words in negative class found by my code includes cancelled, delayed, hours, hold etc which should be in negative tweets. Similar is the case for other two classes. However, one must interpret these result in such ways that most occuring words by all three classes are tags and then the normal words follow. Since we do not discriminate between these tags and words, the results might be misleading as some tags appear in tweets irrespective of the classes due to tweetingh style/behaviour of airline customers. So, a data scientist must take into account these factors.

## References

1. Twitter Airline Sentiment dataset.
https://www.kaggle.com/crowdflower/twitter-airline-sentiment

2. ”On Discriminative vs. Generative Classifiers: A comparison of logistic regression and Naive Bayes” by Andrew Ng and Michael I. Jordan.

3. Manning, C. D., Raghavan, P., and Schutze, H. (2008). Introduction to information retrieval. New York: Cambridge University Press.

http://nlp.stanford.edu/IR-book/html/htmledition/mutual-information-1.html
4. CMU Lecture Notes. http://www.cs.cmu.edu/˜epxing/Class/10701-10s/Lecture/lecture5.pdf

## Appendix

### Python Script:

```
"""
@Author Syed Hasan Raza
CS461 Machine Learning Course
"""

import pandas as pd
import math
import numpy as np

#Loading Necessary files as dataframes using pandas
train_features = pd.read_csv("C:/Users/Hassan/Desktop/Intro to
ML/HW_1/GitHub_Project/Twitter-Sentiment-Analysis/Dataset/question-4-train-features.csv", header = None)
train_labels = pd.read_csv("C:/Users/Hassan/Desktop/Intro to
ML/HW_1/GitHub_Project/Twitter-Sentiment-Analysis/Dataset/question-4-train-labels.csv", header = None)
test_labels = pd.read_csv("C:/Users/Hassan/Desktop/Intro to
ML/HW_1/GitHub_Project/Twitter-Sentiment-Analysis/Dataset/question-4-test-features.csv", header = None)
test_features = pd.read_csv("C:/Users/Hassan/Desktop/Intro to
ML/HW_1/GitHub_Project/Twitter-Sentiment-Analysis/Dataset/question-4-test-labels.csv", header = None)
vocab = pd.read_csv('C:/Users/Hassan/Desktop/Intro to
ML/HW_1/GitHub_Project/Twitter-Sentiment-Analysis/Dataset/question-4-vocab.txt', delimiter = '\t', header =
None)

pos_index = []
neg_index = []
neut_index = []
#This loop saves the indexs of rows of dataframe which correspond to specific tweets by classes
for i in range(len(train_features)):
    if train_labels[0][i] == "positive":
        pos_index.append(i)
    elif train_labels[0][i] == "negative":
        neg_index.append(i)
    else:
        neut_index.append(i)

#calculating prior probabilities by class
prior_pos = len(pos_index)/(len(neg_index)+len(pos_index)+len(neut_index))
prior_neg = len(neg_index)/(len(neg_index)+len(pos_index)+len(neut_index))
prior_neut = len(neut_index)/(len(neg_index)+len(pos_index)+len(neut_index))

#This function returns the number of total words in trainingset for certain class
def total_words_by_class(class_index):
    sum_class = []
    for j in range(len(class_index)):
        sum_class.append(train_features.transpose()[class_index[j]].sum())
    return sum(sum_class)
```

```
total_neut_words = total_words_by_class(neut_index)
total_pos_words = total_words_by_class(pos_index)
total_neg_words = total_words_by_class(neg_index)

#This function returns the likelihoods for multinomial case.
#For Drichlet smoothing, set alpha to 1, otherwise set it to 0
#M stands for multinomial case
def M_likelihood_by_class(class_index, total_class_words, alpha):
    temp = 0
    likelihoods =[]
    for m in range(5722):
        for n in range(len(class_index)):
            temp = temp + train_features[m][class_index[n]]
        likelihoods.append(temp + alpha)
        temp = 0
    return likelihoods/(total_class_words + alpha*5722)

#Calculating thetas for all classes i.e neutral, positive and negative
theta_mle_neut_M = M_likelihood_by_class(neut_index, total_neut_words,0)
theta_mle_pos_M = M_likelihood_by_class(pos_index, total_pos_words,0)
theta_mle_neg_M = M_likelihood_by_class(neg_index, total_neg_words,0)

#Definition of log funtion to include valid answer for log(0) i.e. negative infinity
def log_10 (num):
    if num != 0:
        return math.log10(num)
    else:
        return -math.inf
#This function calculates posterior probabilities for specific class using likelihoods and priors for
Multinomial case
def M_posterior_by_class(M_theta_mle_by_class, prior_class):
    post_class = 0
    posteriors = []
    for s in range(2928):
        for t in range(5722):
            if ((test_features[t][s] != 0) or (M_theta_mle_by_class[t] != 0)):
                post_class = post_class + test_features[t][s]*log_10(M_theta_mle_by_class[t])
        posteriors.append(post_class + math.log10(prior_class))
        post_class = 0
    return posteriors

#Calculating posterior probabilities for all classes i.e neutral, positive and negative
posterior_neut_M = M_posterior_by_class(theta_mle_neut_M, prior_neut)
posterior_pos_M = M_posterior_by_class(theta_mle_pos_M, prior_pos)
posterior_neg_M = M_posterior_by_class(theta_mle_neg_M, prior_neg)

#This function is used for prediction of new tweets (test_set) and returns accuracy and number of wrong
predictions
def prediction_and_accuracy(posterior_neut,posterior_pos, posterior_neg):
    predictions = []
```

```
    true_predict_count = 0

    for y in range(2928):
        m = max(posterior_neut[y],posterior_pos[y],posterior_neg[y])
        if (m == posterior_neut[y]):
            predictions.append("neutral")
        elif (m == posterior_neg[y]):
            predictions.append("negative")
        else:
            predictions.append("positive")

    for z in range(2928):
        if predictions[z] == test_labels[0][z]:
            true_predict_count = true_predict_count + 1

    wrong_predictions = 2928 - true_predict_count
    accuracy = (true_predict_count/2928)*100
    return accuracy, wrong_predictions
#Calculating accuracy and number of wrong preductions for multinomial case on test set
Multinomial_Results = prediction_and_accuracy(posterior_neut_M,posterior_pos_M, posterior_neg_M)
print(Multinomial_Results)

#Drichlet Smoothing from now onwards

#Calculating thetas for all classes i.e neutral, positive and negative in case of Smoothing
smooth_theta_mle_neut = M_likelihood_by_class(neut_index, total_neut_words,1)
smooth_theta_mle_pos = M_likelihood_by_class(pos_index, total_pos_words,1)
smooth_theta_mle_neg = M_likelihood_by_class(neg_index, total_neg_words,1)

#Calculating posterior probabilities for all classes i.e neutral, positive and negative in case of Smoothing
smooth_posterior_neut = M_posterior_by_class(smooth_theta_mle_neut, prior_neut)
smooth_posterior_pos = M_posterior_by_class(smooth_theta_mle_pos, prior_pos)
smooth_posterior_neg = M_posterior_by_class(smooth_theta_mle_neg, prior_neg)

#Calculating accuracy and number of wrong preductions for Drichlet Smoothing case on test set
(Multinomial)
Drichlet_Results = prediction_and_accuracy(smooth_posterior_neut,smooth_posterior_pos,
smooth_posterior_neg)
print(Drichlet_Results)

#Bernoulli part from now onwards

#This function returns the likelihoods for Bernoulli case.
#B stands for Bernoulli case
def B_likelihood_by_class(class_index):
    temp = 0
    likelihoodss =[]
    for m in range(5722):
        for n in range(len(class_index)):
            if train_features[m][class_index[n]] != 0:
```

```
            temp = temp + 1
        likelihoodss.append(temp )
        temp = 0
    return likelihoodss/(np.int64(len(class_index)))

#Calculating thetas for all classes i.e neutral, positive and negative for Bernoulli case
theta_mle_neut_B = B_likelihood_by_class(neut_index)
theta_mle_pos_B = B_likelihood_by_class(pos_index)
theta_mle_neg_B = B_likelihood_by_class(neg_index)

#This function calculates posterior probabilities for specific class using likelihoods and priors for Bernoulli
case
def posterior_by_class(theta_mle_by_class, prior_by_class):
    post_class = 0
    posteriorss = []
    for s in range(2928):
        for t in range(5722):
            if ((test_features[t][s] != 0)):
                post_class = post_class + log_10(theta_mle_by_class[t])
            else:
                post_class = post_class + log_10( 1 - theta_mle_by_class[t])
        posteriorss.append(post_class + math.log10(prior_by_class))
        post_class = 0
    return posteriorss

#Calculating posterior probabilities for all classes i.e neutral, positive and negative for Bernoulli case
posterior_neut_B = posterior_by_class(theta_mle_neut_B, prior_neut)
posterior_pos_B = posterior_by_class(theta_mle_pos_B, prior_pos)
posterior_neg_B = posterior_by_class(theta_mle_neg_B, prior_neg)
#Calculating accuracy and number of wrong preductions for Bernoulli case on test set
Bernoulli_Results = prediction_and_accuracy(posterior_neut_B,posterior_pos_B, posterior_neg_B)
print(Bernoulli_Results)

#Last Part from now onwards to find most common 20 words by class

#This function finds the most common 20 words according to given class
def common_20(class_index):
    count = 0
    word_occurences =[]
    common_words_20 = []
    for m in range(5722):
        for n in range(len(class_index)):
            count = count + train_features[m][class_index[n]]
        word_occurences.append(count)
        count = 0
    word_occurences_sorted = np.flip(np.argsort(word_occurences))
    for i in range(20):
        common_words_20.append(vocab[0][word_occurences_sorted[i]])
    return common_words_20
```

```
#calculating most common 20 words for each class
common_words_20_pos = common_20(pos_index)
print(common_words_20_pos)
common_words_20_neg = common_20(neg_index)
print(common_words_20_neg)
common_words_20_neut = common_20(neut_index)
print(common_words_20_neut)
```