Twitter Sentiment Analysis:

Sentiment analysis is the automated process of identifying and classifying subjective information in text data. This might be an opinion, a judgment, or a feeling about a particular topic or product feature. The most common type of sentiment analysis is 'polarity detection' and involves classifying statements as Positive, Negative or Neutral.

Overview of code by cell

Cell 1:
Import all libraries that are used by the code.

Cell 2:
Mounting Google Drive to be able to use other files in the drive.

Cell 3:
Assigning variables for training and testing data.

Cell 4:
Selecting only the tweets with a label value of 0, and print the first 3 rows of that selection. This would show the first 3 tweets in the dataset that have a positive sentiment, as indicated by the label value of 0.

Cell 5:
Selecting only the tweets with a label value of 1, and print the first 3 rows of that selection. This would show the first 3 tweets in the dataset that have a negative sentiment, as indicated by the label value of 1.

Cell 6:
Plotting the counts of positive and negative sentiment tweets of the training dataset using the matplotlib library.

Cell 7:
Adding a length attribute to the training and test dataset.

Cell 8:
Outputting the statistical information of both the sentiment labels.

Cell 9:
Plotting a histogram of the length of a tweet and the frequency of the different lengths of a tweet using the matplotlib library.

Cell 10:
Loads the tweet dataset into a dataframe called train, create a CountVectorizer object, fit it to the tweet column of the train dataframe, and then use it to count the number of times each word appears in the dataset. It would then create a dataframe of the top 20 words by frequency and plot the results as a bar chart.

Cell 11:
Loads the tweet dataset into a dataframe called train, create a list of (word, count) tuples sorted in descending order by count, create a dictionary of word frequencies from that list, create a WordCloud object, and generate a word cloud from the word frequencies, and finally create a figure and plot the word cloud on it.

Cell 12:
Loads the tweet datasets with label 0 (positive words), create a list of (word, count) tuples sorted in descending order by count, create a dictionary of word frequencies from that list, create a WordCloud object, and generate a word cloud of positive words from the word frequencies, and finally create a figure and plot the word cloud on it.

Cell 13:
Loads the tweet datasets with label 1 (negative words), create a list of (word, count) tuples sorted in descending order by count, create a dictionary of word frequencies from that list, create a WordCloud object, and generate a word cloud of negative words from the word frequencies, and finally create a figure and plot the word cloud on it.

Cell 14:
A helper function that creates a list of tweets after it extracts the hashtags from the tweets using regex. The output would be a list of lists of hashtags, where each sublist contains the hashtags from a single tweet.

Cell 15:
In this code, ht_positive is a list of lists of hashtags that appear in tweets with a positive sentiment, and the label column is used to indicate the sentiment of each tweet.
The hashtag_extract function is called on the tweet column of the train dataframe, but only for tweets with a label value of 0. The resulting list of lists of hashtags is then stored in the ht_positive variable. The ht_positive list is then flattened using the sum function and an empty list as an argument. This creates a single list containing all of the hashtags from the tweets with a positive sentiment, with no nested lists. The nltk.FreqDist function is then used to create a frequency distribution of the hashtags in the ht_positive list. This function returns a dictionary-like object where the keys are the hashtags and the values are their corresponding

frequencies. A data frame is then created from the FreqDist object, with columns named Hashtag and Count. The Hashtag column contains the hashtags, and the Count column contains their corresponding frequencies. The first 10 rows of the dataframe are then selected using the nlargest method, and the resulting data is plotted as a bar chart.

Cell 16:
Identical to Cell 15, but with negative sentiment or label value of 1.

Cell 17:
In this cell, tokenized_tweet is a list of lists of words that are extracted from the tweet column of the train dataframe. The model_w2v variable is then assigned to a Word2Vec object from the gensim library. This object is being initialized with several parameters that specify the properties of the word embeddings that will be generated. Then the train method trains the Word2Vec model on the tokenized tweets, generating word embeddings that capture the relationships between the words in the dataset.

Cell 18:
model_w2v.wv.most_similar is a method that is part of the Word2Vec class in the gensim library. This method is used to find the words in the trained word embedding model that are most similar to the word specified in the positive/negative parameter. In this cell this method is called with the parameter of a positive sentiment with the word “computer,” and would return a list of the words that are most similar to the word "computer" in the context of the tweets in the dataset.

Cell 19:
Identical to 18, but the parameter is positive = “food.”

Cell 20:
Identical to 18, but the parameter is negative = “computer.”

Cell 21:
Identical to 18, but the parameter is negative = “food.”

Cell 22:
In this cell, add_label is the name of a function that takes a list of lists of words and returns a list of LabeledSentence objects from the gensim library. We then run this function using the tokenized_tweet from cell 17 as the parameter and return the last two LabeledSentence objects from the returned list.

Cell 23:

Downloading package “stopwords.”

Cell 24:
Processing all the tweets in the training data. All words become lower case and are made into lists. They are then processed by PorterStemmer, which is in the nltk library. This allows similar variations of the same word to be processed as the same. A list comprehension is then used to remove any words in the list of words that are also present in the list of stopwords (common words that are often removed from text data to improve the performance of natural language processing models).

Cell 25:
Identical to 24 with the test data.

Cell 26:
The CountVectorizer object is initialized with only the 2500 most frequent words in the text data and will be used to create the matrix of token counts. The fit_transform method is then called on the cv object and passed the train_corpus list as an argument. This method learns the vocabulary of the text data and then creates a matrix of token counts, where each row represents a document (in this case, a tweet) and each column represents a word. The resulting matrix is then converted to a NumPy array stored as “x”.The “y” variable is then assigned to the label column of the train dataframe. Then train_test_split is used to reserve 25% of the data for validation and 75% for training. The StandardScaler class from the sklearn.preprocessing module is then used to standardize the features in the x_train, x_valid, and x_test arrays to improve performance.

Cell 27:
The RandomForestClassifier object is first created and then fit to the training data (x_train and y_train) using the fit method. This method trains the model on the training data and learns the relationships between the features and the labels. Once the model has been trained, the predict method is called on the RandomForestClassifier object and passed the validation set features (x_valid) as an argument. This method makes predictions on the validation set using the trained model. The predictions are stored in the y_pred variable. Next, the score method is called on the RandomForestClassifier object and computes the accuracy of the model on the training and validation sets, which is the proportion of correct predictions made by the model.

Cell 28:
Identical to 27 with the LogisticRegression object.

Cell 29:
Identical to 27 with the DecisionTreeClassifier object.

Cell 30:
Identical to 27 with the SVC object.

Documentation of usages of APIs

The libraries we used are numpy, pandas, regex, nltk, matplotlib, and seaborn. We use numpy overall to do work with arrays. We use pandas to allow for data analysis of the tweets. We use the regex library to deal with characters, which we specifically do to determine the hashtags in the tweets (denoted by the pound symbol). The nltk (natural language toolkit) allows us to use natural language processing to analyze the words in the tweet. Matplotlib is the api which allows us to do mathematical analysis on the data we gather. Seaborn is the library which allows us to create visuals and graphs based on the data we have gathered.

Instructions to install software

Running the notebook is all that is needed. All installations are imports on the notebook in the first cell.

Brief description of contribution of each team member

Aadarsh Krishnan: Wrote code and wrote documentation.
Minal Singh: Researched for data sets, wrote code, and made presentations.
Neil Sahai: Wrote code and made/presented presentation
Param Mehta: Wrote code and made/presented presentation