**Economics 1680: Machine Learning, Text Analysis, and Economics**

Brown University

Prof. Amy Handlan

This assignment will cover applications of text analysis regarding clustering and classification. For this assignment, you should write/type your answers into this worksheet. You must also submit your python code. Alternatively, you can make a Jupyter notebook and combine your written answers, code, and output. You need to submit both to Canvas and Github.

You may discuss the problem set with your class mates, but every student must do their own work. It is always important to cite our references that help us in our work. Please cite the students you work with here: \_\_\_ , \_\_\_\_, \_\_\_\_\_

**PACKAGES TO INSTALL**

**If you are using Anaconda:**

conda install -c conda-forge transformers

**If you are using Pip:**

pip install -q transformers

**PACKAGES TO IMPORT:**

﻿import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from collections import Counter

from itertools import chain

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from fuzzywuzzy import fuzz

from sklearn.linear\_model import LogisticRegression

from transformers import pipeline

1. **CODING EFFICIENCY AND COMMENTING (10 POINTS)**

10 points will be given according to the efficiency and cleanliness of your code. It should produce output that answers the questions from the homework only and everything should be commented. Furthermore, your writeup and code should be submitted both to Canvas and Github. These points are meant to encourage and reward good coding habits.

1. **DOWNLOAD DATA, SUMMARIZE, AND PROCESS TEXT (15 POINTS)**
2. Download data on covid tweets from Kaggle: <https://www.kaggle.com/datasets/datatattle/covid-19-nlp-text-classification/download>

Load both the training and testing files into python.

1. Plot histogram of sentiment categories in the *training* dataframe. Hint: Use df[‘Sentiment’] ﻿.value\_counts().reindex(["Extremely Negative", "Negative", "Neutral", "Positive", "Extremely Positive"]).plot(kind='bar') to control ordering of categories. Be sure to add title and labels for your axes. Insert your graph here.
2. Plot histogram of sentiment categories in the *testing* dataframe. Insert your graph here.
3. Compare the distributions. Do you think this is a good training/testing split? Why or why not? Write your answer here in 2-3 sentences.
4. For this homework, you will be working with the testing dataset only because it is a smaller sample. Furthermore, drop all Neutral labelled tweets. Call this dataframe: df\_tweet. How many observations (tweets) does this new dataframe have? Write your answer here.
5. **PROCESSING AND CLEANING TEXT (20 POINTS)**
6. To clean the tweets a little, run the following code. NOTE: this code will take between 5-10 minutes to run on a laptop. Explain what each line is doing:

﻿cleantweet=[]

for i in range(len(df\_tweet)):

tweet=df\_tweet['OriginalTweet'][i]

tweet=tweet.replace('\r',' ')

tweet=tweet.replace('\n',' ')

tweet=tweet.replace('. ',' ')

tweet=tweet.replace(', ',' ')

tokens = tweet.split(' ')

tweet\_hashtags=[]

tweet\_token=[]

for t in tokens:

if "https:" not in t:

tweet\_token.append(t)

cleantweet.append(" ".join(tweet\_token).lower())

df\_tweet['clean tweet']= cleantweet

#### Below Takes a few min to run. 5-10 min. ####

similar\_level=85

duplicate\_index = []

for original in range(len(df\_tweet)-1):

if original not in duplicate\_index:

for compared in range(original+1, len(df\_tweet)):

if compared not in duplicate\_index:

if fuzz.ratio(df\_tweet['clean tweet'][original], df\_tweet['clean tweet'][compared]) >= similar\_level:

duplicate\_index.append(compared)

df\_tweet.drop(duplicate\_index, inplace=True)

df\_tweet.reset\_index(drop=True,inplace=True)

1. Make a list of hashtags for each tweet.
   1. Loop over rows in df\_tweet
   2. Split each tweet by spaces using .split(‘ ‘)
   3. Make a list of tokens with # in them. Hint: make a list of lists.
2. What are the top 20 most used hashtags? Insert the list of hashtags and how many time each hashtag occur here. Hint: use ﻿Counter(chain.from\_iterable()) and .most\_common().
3. Using the ‘clean tweet’ variable from problem 6, create term frequency vector representation for all tweets. That is, make a document-term matrix where each document is a tweet and the terms are all the vocabulary from the tweet dataset. Hint: use TfidfVectorizer().fit\_transform(). Call your document term matrix X. It should come out as a ﻿scipy.sparse.csr.csr\_matrix object.
   1. What are the dimensions of your matrix? Write that out here.
4. Use .get\_feature\_names() to make a list of the vocabulary and call it “features”.
   1. Is this list alphabetized automatically? Write your answer here:
   2. What is the word associated with index 1680? Write your answer here:
5. **SIMILARITY MEASURES (10 POINTS)**
6. Calculate the pairwise similarity between all tweets using cosine similarity. Hint: use the cosine\_similarity() command from sklearn and then change it to a dataframe called df\_cossim.
7. Run the following code and explain each step:

﻿df\_cossim\_tri = pd.DataFrame(np.tril(df\_cossim.values, k=-1))

df\_rank = df\_cossim\_tri.unstack().reset\_index()

df\_rank.columns = ['row', 'column', 'similarity']

1. Which pairs of tweets have the top 10 highest similarity scores? Below include a table with the tweet pairs and their corresponding similarity score. Hint: sort the df\_rank dataframe over similarity.
2. Spend 3-5 sentences describing what seems to make sense in the similarity matching. What doesn’t make sense? If you wanted to improve matching, what steps would you take now? If you would need to do more cleaning of the data, what would you want to do?
3. **LOGIT-LASSO REGRESSION (25 POINTS)**
4. Make a new variable that has binary sentiment (positive=1 and negative=0) where a sentiment of positive and extremely positive is now 1, and then negative and extremely negative is now 0.
5. Run a logit lasso regression with binary sentiment as the output variable and your document term matrix as your inputs.
   1. Convert your binary sentiment column to an array called y, and your document term matrix to an array called x. Hint: use X.toarray() and .to\_numpy() for your df column.
   2. Use the logit regression command with an L1 penalty, liblinear solver, and random state of 1680. Loop over the hyperparameter “C” which controls the weight in front of the penalty by considering values of C in ﻿ [0.001, 0.01, 0.1, 1, 10, 100, 1000]. Hint: model.score(x,y) will tell you the prediction accuracy for the logit model fit to x,y. Pick the C value that has the maximum accuracy.
   3. Save the model coefficients as “coefs”. Hint: use model.coef\_
   4. Run the following code and explain each line

﻿df\_coef = pd.DataFrame({'word': features, 'coef': coefs[-1].tolist()})

df\_coef['abs coef']= df\_coef['coef'].abs()

df\_tweet['logitlasso predicted sentiment'] = model.predict(x)

1. What are the top 20 most important words for predicting class? These will be the largest coefficients in absolute value. Hint: sort df\_coef. Print those top 20 words and their coefficients here.
2. What are the top 10 “positive words”? The top 10 “negative words”? Print the top 10 words of each sentiment and their coefficients here.
3. **PRETRAINED MACHINE LEARNING ALGORITHM (HUGGING FACE) (20 POINTS)**
4. Here you will use the HuggingFace package with pre-trained algorithms. This pipeline takes string text as input, not tokens. So you will be using your dataframe from question 5. First work with the default model for sentiment analysis from HuggingFace which is the DistilBERT base uncased model finetuned with SST-2.
   1. Run the following code

sentiment\_pipeline = pipeline("sentiment-analysis")

print(sentiment\_pipeline(df\_tweet['OriginalTweet'][0:10].tolist()))

print(﻿df\_tweet[‘OriginalTweet’][0:10].tolist())

* 1. Paste the output from the code here:
  2. How does this compare to the original sentiment label? Print out the original sentiment label for the first 10 tweets.
  3. Do you think the hand coded labels (the original sentiment labels) or the ones predicted by the pre-trained algorithm make more sense? Explain your answer using the text from the tweets as evidence. Why do you think there are differences?

1. Look at the other pretrained models in HuggingFace at https://huggingface.co/models?pipeline\_tag=text-classification&sort=downloads&search=sentiment.
   1. Which pretrained models would be appropriate for analyzing the Covid Tweets dataset? Name the model you would use, broadly how it was trained, and your reasoning for why you would pick it.
   2. Take the code from 19a. and run the sentiment analysis for the same 10 tweets. Print the predicted labels here:
   3. How do the predicted labels here compare to those from question 19? Explain why you think there are differences/similarities.