import pandas as pd

import numpy as np

from sklearn import model\_selection, preprocessing, linear\_model, naive\_bayes, metrics, svm

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn import decomposition, ensemble

import pandas, xgboost, numpy, textblob, string

from keras.preprocessing import text, sequence

from keras import layers, models, optimizers

import tensorflow

tensorflow.keras.\_\_version\_\_

apolitcal = pd.read\_csv('articles\_copy.csv', encoding='cp1252')

apolitcal.head()

apolitcal.shape

test = apolitcal.iloc[3]

test

j = apolitcal['article']

j[1]

X = apolitcal.drop("political leaning", axis=1)

y = apolitcal["political leaning"]

print(X.shape, y.shape)

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from tensorflow.keras.utils import to\_categorical

# split the dataset into training and validation datasets

train\_x, valid\_x, train\_y, valid\_y = model\_selection.train\_test\_split(apolitcal['political leaning'], apolitcal['article'])

# label encode the target variable

encoder = preprocessing.LabelEncoder()

train\_y = encoder.fit\_transform(train\_y)

valid\_y = encoder.fit\_transform(valid\_y)

# create a count vectorizer object

count\_vect = CountVectorizer(analyzer='word', token\_pattern=r'\w{1,}')

count\_vect.fit(apolitcal['article'])

# transform the training and validation data using count vectorizer object

xtrain\_count = count\_vect.transform(train\_x)

xvalid\_count = count\_vect.transform(valid\_x)

# word level tf-idf

tfidf\_vect = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', max\_features=5000)

tfidf\_vect.fit(apolitcal['article'])

xtrain\_tfidf = tfidf\_vect.transform(train\_x)

xvalid\_tfidf = tfidf\_vect.transform(valid\_x)

# ngram level tf-idf

tfidf\_vect\_ngram = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram.fit(apolitcal['article'])

xtrain\_tfidf\_ngram = tfidf\_vect\_ngram.transform(train\_x)

xvalid\_tfidf\_ngram = tfidf\_vect\_ngram.transform(valid\_x)

# characters level tf-idf

tfidf\_vect\_ngram\_chars = TfidfVectorizer(analyzer='char', token\_pattern=r'\w{1,}', ngram\_range=(2,3), max\_features=5000)

tfidf\_vect\_ngram\_chars.fit(apolitcal['article'])

xtrain\_tfidf\_ngram\_chars = tfidf\_vect\_ngram\_chars.transform(train\_x)

xvalid\_tfidf\_ngram\_chars = tfidf\_vect\_ngram\_chars.transform(valid\_x)

apolitcal['char\_count'] = apolitcal['article'].apply(len)

apolitcal['word\_count'] = apolitcal['article'].apply(lambda x: len(x.split()))

apolitcal['word\_density'] = apolitcal['char\_count'] / (apolitcal['word\_count']+1)

apolitcal['punctuation\_count'] = apolitcal['article'].apply(lambda x: len("".join(\_ for \_ in x if \_ in string.punctuation)))

apolitcal['title\_word\_count'] = apolitcal['article'].apply(lambda x: len([wrd for wrd in x.split() if wrd.istitle()]))

apolitcal['upper\_case\_word\_count'] = apolitcal['article'].apply(lambda x: len([wrd for wrd in x.split() if wrd.isupper()]))

testing = apolitcal['word\_count']

testing

def train\_model(classifier, feature\_vector\_train, label, feature\_vector\_valid, is\_neural\_net=False):

# fit the training dataset on the classifier

classifier.fit(feature\_vector\_train, label)

# predict the labels on validation dataset

predictions = classifier.predict(feature\_vector\_valid)

if is\_neural\_net:

predictions = predictions.argmax(axis=-1)

return metrics.accuracy\_score(predictions, valid\_y)

# Naive Bayes on Count Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_count, train\_y, xvalid\_count)

print ("NB, Count Vectors: ", accuracy)

# Naive Bayes on Word Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf, train\_y, xvalid\_tfidf)

print ("NB, WordLevel TF-IDF: ", accuracy)

# Naive Bayes on Ngram Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram, train\_y, xvalid\_tfidf\_ngram)

print ("NB, N-Gram Vectors: ", accuracy)

# Naive Bayes on Character Level TF IDF Vectors

accuracy = train\_model(naive\_bayes.MultinomialNB(), xtrain\_tfidf\_ngram\_chars, train\_y, xvalid\_tfidf\_ngram\_chars)

print ("NB, CharLevel Vectors: ", accuracy)

#Boosting Models

# Extereme Gradient Boosting on Count Vectors

#accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_count.tocsc(), train\_y, xvalid\_count.tocsc())

#print ("Xgb, Count Vectors: ", accuracy)

# Extereme Gradient Boosting on Word Level TF IDF Vectors

#accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_tfidf.tocsc(), train\_y, xvalid\_tfidf.tocsc())

#print ("Xgb, WordLevel TF-IDF: ", accuracy)

# Extereme Gradient Boosting on Character Level TF IDF Vectors

#accuracy = train\_model(xgboost.XGBClassifier(), xtrain\_tfidf\_ngram\_chars.tocsc(), train\_y, xvalid\_tfidf\_ngram\_chars.tocsc())

#print ("Xgb, CharLevel Vectors: ", accuracy)

# download nltk stuff

import nltk

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')

# methods to clean up articles

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

from nltk import word\_tokenize

import string

import re

stop\_words\_ = set(stopwords.words('english'))

#stopwords = nltk.corpus.stopwords.words('english')

#to add words to remove

stop\_words\_.update(('mr','m','said','u','and','I','A','And','So','arnt','This','When','It','many','Many','so','cant','Yes','yes','No','no','These','these'))

wn = WordNetLemmatizer()

def accept\_words(token):

return token not in stop\_words\_ and token not in list(string.punctuation)

def clean\_txt(text):

clean\_text = []

text = re.sub("'", "",text)

text=re.sub("[\d\W]+"," ",text)

clean\_text = [ wn.lemmatize(word) for word in word\_tokenize(text.lower()) if accept\_words(word)]

return " ".join(clean\_text)

stop\_words\_

#!pip install textstat

# additional features (subjectivity and polarity)

# however, have not used polarity since naive bayes can't take negative values

from textblob import TextBlob

from sklearn.feature\_extraction import DictVectorizer

import textstat

def hard\_words(text):

total\_words = len(text.split())

return textstat.difficult\_words(text)/total\_words

def subj\_txt(text):

return TextBlob(text).sentiment[1]

def polarity\_txt(text):

return TextBlob(text).sentiment[0]

#home works

#COMPARISON OF SUB WORDS FOR CONSERVATIVES VS LIBERALS

#COMPARISON OF PRO WORDS FOR CONSERVATIVES VS LIBERALS

#COMPARISON OF HARD WORDS FOR CONSERVATIVES VS LIBERALS

apolitcal['subj'] = apolitcal['cleaned'].apply(subj\_txt)

apolitcal['pol'] = apolitcal['cleaned'].apply(polarity\_txt)

apolitcal['difficult\_words'] = apolitcal['cleaned'].apply(hard\_words)

apolitcal.head()

apolitcal\_sub = apolitcal.groupby('political leaning')['subj']

apolitcal\_sub.aggregate('mean')

df\_apolitcal\_sub = pd.DataFrame(apolitcal\_sub.aggregate('mean'))

df\_apolitcal\_sub

df\_sub\_c = df\_apolitcal\_sub.iloc[0, 0]

df\_sub\_l = df\_apolitcal\_sub.iloc[1, 0]

apolitcal\_pol = apolitcal.groupby('political leaning')['pol']

apolitcal\_pol.aggregate('mean')

df\_apolitcal\_pol = pd.DataFrame(apolitcal\_pol.aggregate('mean'))

df\_apolitcal\_pol

df\_pol\_c = df\_apolitcal\_pol.iloc[0, 0]

df\_pol\_l = df\_apolitcal\_pol.iloc[1, 0]

apolitcal\_dw = apolitcal.groupby('political leaning')['difficult\_words']

apolitcal\_dw.aggregate('mean')

df\_apolitcal\_dw = pd.DataFrame(apolitcal\_dw.aggregate('mean'))

df\_apolitcal\_dw

df\_dw\_c = df\_apolitcal\_dw.iloc[0, 0]

df\_dw\_c

df\_dw\_l = df\_apolitcal\_dw.iloc[1, 0]

df\_dw\_l

apolitcal.groupby(by='political leaning')['difficult\_words'].mean()

apolitcal.query("political leaning"=='conservative')['difficult\_words'].mean()

apolitcal.loc[(apolitcal['political leaning'] == 'liberal')].sum()

import matplotlib.pyplot as plt

import numpy as np

np.random.seed(19680801)

plt.rcdefaults()

fig, ax = plt.subplots()

# Example data

people = apolitcal\_sub

#y\_pos = np.arange(len(people))

performance = apolitcal\_sub.aggregate('mean')

#error = np.random.rand(len(people))

ax.barh(y\_pos, performance, xerr=error, align='center')

#ax.set\_yticks(y\_pos)

#ax.set\_yticklabels(people)

#ax.invert\_yaxis() # labels read top-to-bottom

ax.set\_xlabel('Political views')

ax.set\_title('How vocal do you want to go today?')

plt.show()

import plotly.express as px

df = px.data.tips()

fig = px.scatter(df, x="total\_bill", y="tip", facet\_col="smoker", color="sex", trendline="ols")

fig.show()

results = px.get\_trendline\_results(fig)

print(results)

results.query("sex == 'Male' and smoker == 'Yes'").px\_fit\_results.iloc[0].summary()

import plotly.graph\_objects as go

animals=['Difficult words', 'Subjective Text', 'Polarized Text']

fig = go.Figure(data=[

go.Bar(name='Liberal', x=animals, y=[df\_dw\_l, df\_sub\_l, df\_pol\_l]),

go.Bar(name='Conservative', x=animals, y=[df\_dw\_c, df\_sub\_c, df\_pol\_c])

])

# Change the bar mode

fig.update\_layout(barmode='stack')

fig.show()

import plotly.express as px

df = px.data.tips()

fig = px.bar(apolitcal, x="word\_count", y="difficult\_words", color="word\_count", barmode="group",

facet\_row="political leaning", facet\_col="article",

category\_orders={"article": ["difficult\_words", "subj", "pol"],

"political leaning": ["liberal", "conservative"]})

fig.show()

apolitcal['article'][0]

clean\_txt(apolitcal['article'][0])

#Word Count

%matplotlib inline

import scattertext as st

import re, io

from pprint import pprint

import pandas as pd

import numpy as np

from scipy.stats import rankdata, hmean, norm

import spacy

import os, pkgutil, json, urllib

from urllib.request import urlopen

from IPython.display import IFrame

from IPython.core.display import display, HTML

from scattertext import CorpusFromPandas, produce\_scattertext\_explorer

display(HTML("&lt;style>.container { width:98% !important; }&lt;/style>"))

import en\_core\_web\_sm

nlp = en\_core\_web\_sm.load()

doc = nlp("This is a sentence.")

doc

apolitcal.head(20)

apolitcal['cleaned'] = apolitcal['article']

apolitcal.head()

count = 0

while (count < len(apolitcal)):

apolitcal['cleaned'][count] = clean\_txt(apolitcal['article'][count])

count = count + 1

print (count)

else:

print("In Else Block")

apolitcal['parsed'] = apolitcal.cleaned.apply(nlp)

apolitcal.head()

print("Document Count")

print(apolitcal.groupby('political leaning')['parsed'].count())

print("Word Count")

apolitcal.groupby('political leaning').apply(lambda x: x.article.apply(lambda x: len(x.split())).sum())

corpus = st.CorpusFromParsedDocuments(apolitcal, category\_col='political leaning', parsed\_col='parsed').build()

corpus

#This shows the Top 10 most frequent used words by Republican and Democrats

#candidates and how frequently each words are used per 25k words and what’s it’s tf-idf score

html = produce\_scattertext\_explorer(corpus,

category='liberal',

category\_name='Liberal',

not\_category\_name='Conservative',

width\_in\_pixels=1000,

minimum\_term\_frequency=5,

transform=st.Scalers.scale,

metadata=apolitcal['Source'])

file\_name = 'output/Conventions2019ScattertextScale.html'

open(file\_name, 'wb').write(html.encode('utf-8'))

IFrame(src=file\_name, width = 1200, height=700)

#This shows the characteristic terms used by each source

#from both parties and which is more informative compared to the graph above

html = st.produce\_scattertext\_explorer(corpus,

category='liberal',

category\_name='Liberal',

not\_category\_name='Conservative',

minimum\_term\_frequency=5,

width\_in\_pixels=1000,

transform=st.Scalers.log\_scale\_standardize)

file\_name = 'output/Conventions2019ScattertextLog.html'

open(file\_name, 'wb').write(html.encode('utf-8'))

IFrame(src=file\_name, width = 1200, height=700)

#The follow visuals ranks each term by frequency percentile instead of raw frequencies

html = produce\_scattertext\_explorer(corpus,

category='liberal',

category\_name='Liberal',

not\_category\_name='Conservative',

width\_in\_pixels=1000,

minimum\_term\_frequency=5,

transform=st.Scalers.percentile,

metadata=apolitcal['Source'])

file\_name = 'output/Conventions2019ScattertextRankData.html'

open(file\_name, 'wb').write(html.encode('utf-8'))

IFrame(src=file\_name, width = 1200, height=700)