import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import re

import string

import nltk

from nltk.corpus import wordnet

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import seaborn as sns

import plotly.express as px

import matplotlib.pyplot as plt

from wordcloud import WordCloud

import plotly.graph\_objects as go

from sklearn import metrics

from sklearn.metrics import ConfusionMatrixDisplay,confusion\_matrix

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

from sklearn.feature\_extraction.text import CountVectorizer

df=pd.read\_csv("/content/drive/MyDrive/SENTIMENT.csv",encoding='cp1252')

print(df)

df = df.drop(['summary'], axis=1)

df["rating"] = df["rating"].apply(lambda x: 1 if x < 3  else 0)

df

 df["reviewText"] = df["reviewText"].str.lower()

df.head()

PUNCT\_TO\_REMOVE = string.punctuation

def remove\_punctuation(text):

    return text.translate(str.maketrans('', '', PUNCT\_TO\_REMOVE))

df["reviewText"] = df["reviewText"].apply(lambda text: remove\_punctuation(text))

df.head()

import nltk

nltk.download('stopwords')

STOPWORDS = set(stopwords.words('english'))

def remove\_stopwords(text):

    """custom function to remove the stopwords"""

    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df["reviewText"] = df["reviewText"].apply(lambda text: remove\_stopwords(text))

df.head()

import nltk

nltk.download('wordnet')

import nltk

nltk.download('omw-1.4')

import nltk

nltk.download('averaged\_perceptron\_tagger')

lemmatizer = WordNetLemmatizer()

wordnet\_map = {"N":wordnet.NOUN, "V":wordnet.VERB, "J":wordnet.ADJ, "R":wordnet.ADV}

def lemmatize\_words(text):

    pos\_tagged\_text = nltk.pos\_tag(text.split())

    return " ".join([lemmatizer.lemmatize(word, wordnet\_map.get(pos[0], wordnet.NOUN)) for word, pos in pos\_tagged\_text])

df["reviewText"] = df["reviewText"].apply(lambda text: lemmatize\_words(text))

df.head()

train, test = train\_test\_split(df, test\_size = 0.3, stratify = df['rating'], random\_state = 42)

cv= CountVectorizer(binary=True, min\_df = 10, max\_df = 0.95)

cv.fit\_transform(train['reviewText'].values)

train\_feature\_set=cv.transform(train['reviewText'].values)

test\_feature\_set=cv.transform(test['reviewText'].values)

feature\_cols = ['helpful', 'rating', 'reviewText','reviewTime','reviewerID','reviewerName','unixReviewTime']

X = df[feature\_cols]

y = df.rating

 pip install sklearn

from sklearn import svm

cls = svm.SVC(kernel="rbf", random\_state=1, gamma=0.03, C=1.5)

y\_train = train['rating'].values

y\_test = test['rating'].values

cls.fit(train\_feature\_set,y\_train)

y\_pred = cls.predict(test\_feature\_set)

cm1 = confusion\_matrix(y\_test, y\_pred)

cm1

disp = ConfusionMatrixDisplay(confusion\_matrix=cm1,display\_labels=cls.classes\_)

disp.plot()

plt.show()

tp=cm1[0][0]

tn=cm1[1][1]

fp=cm1[0][1]

fn=cm1[1][0]

Accuracy=round(metrics.accuracy\_score(y\_test,y\_pred),3)

print("Accuracy=",Accuracy\*100)

precision=tp/(tp+fp)

print("Precision =",precision\*100)

recall=tp/(tp+fn)

print("Recall =",recall\*100)

fm=2\*precision\*recall/(precision+recall)

print("F measure=",fm\*100)

text = " ".join(review for review in df.reviewText)

wordcloud = WordCloud(stopwords=STOPWORDS, background\_color="white").generate(text)

plt.figure( figsize=(10,5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

def term\_frequency(df):

    tf1 = (df['reviewText'][1:2]).apply(lambda x: pd.value\_counts(x.split(" "))).sum(axis = 0).reset\_index()

    tf1.columns = ['words','tf']

    return tf1.head()

term\_frequency(train)

term\_frequency(test)

tf1 = (train['reviewText'][1:2]).apply(lambda x: pd.value\_counts(x.split(" "))).sum(axis = 0).reset\_index()

tf1.columns = ['words','tf']

tf1.head()

tf2 = (test['reviewText'][1:2]).apply(lambda x: pd.value\_counts(x.split(" "))).sum(axis = 0).reset\_index()

tf2.columns = ['words','tf']

tf2.head()

tf1 = (train['reviewText'][1:2]).apply(lambda x: pd.value\_counts(x.split(" "))).sum(axis = 0).reset\_index()

tf1.columns = ['words','tf']

for i,word in enumerate(tf1['words']):

    tf1.loc[i, 'idf'] = np.log(train.shape[0]/(len(train[train['reviewText'].str.contains(word)])))

tf1['tfidf'] = tf1['tf'] \* tf1['idf']

tf1

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split769

from sklearn.metrics import roc\_auc\_score

from keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from keras.utils.np\_utils import to\_categorical

from sklearn import metrics

from sklearn.metrics import ConfusionMatrixDisplay,confusion\_matrix

df=pd.read\_csv("/content/drive/MyDrive/SENTIMENT.csv")

print(df)

df.head()

plt.hist(df['rating'])

df['target'] = df['rating']<4

df

train\_text, test\_text, train\_y, test\_y = train\_test\_split(df['reviewText'],df['target'],test\_size = 0.2)

train\_text.shape

MAX\_NB\_WORDS = 20000

texts\_train = train\_text.astype(str)

texts\_test = test\_text.astype(str)

tokenizer = Tokenizer(nb\_words=MAX\_NB\_WORDS, char\_level=False)

tokenizer.fit\_on\_texts(texts\_train)

sequences = tokenizer.texts\_to\_sequences(texts\_train)

sequences\_test = tokenizer.texts\_to\_sequences(texts\_test)

word\_index = tokenizer.word\_index

print('Found %s unique tokens.' % len(word\_index))

sequences[0]

type(tokenizer.word\_index), len(tokenizer.word\_index)

index\_to\_word = dict((i, w) for w, i in tokenizer.word\_index.items())

" ".join([index\_to\_word[i] for i in sequences[0]])

seq\_lens = [len(s) for s in sequences]

print("average length: %0.1f" % np.mean(seq\_lens))

print("max length: %d" % max(seq\_lens))

%matplotlib inline

import matplotlib.pyplot as plt

plt.hist(seq\_lens, bins=50);

plt.hist([l for l in seq\_lens if l < 200], bins=50);

MAX\_SEQUENCE\_LENGTH = 150

x\_train = pad\_sequences(sequences, maxlen=MAX\_SEQUENCE\_LENGTH)

x\_test = pad\_sequences(sequences\_test, maxlen=MAX\_SEQUENCE\_LENGTH)

print('Shape of data tensor:', x\_train.shape)

print('Shape of data test tensor:', x\_test.shape)

y\_train = train\_y

y\_test = test\_y

y\_train = to\_categorical(np.asarray(y\_train))

print('Shape of label tensor:', y\_train.shape)

from keras.layers import Dense, Input, Flatten

from keras.layers import GlobalAveragePooling1D, Embedding

from keras.models import Model

EMBEDDING\_DIM = 50

N\_CLASSES = 2

sequence\_input = Input(shape=(MAX\_SEQUENCE\_LENGTH,), dtype='int32')

embedding\_layer = Embedding(MAX\_NB\_WORDS, EMBEDDING\_DIM,

                            input\_length=MAX\_SEQUENCE\_LENGTH,

                            trainable=True)

embedded\_sequences = embedding\_layer(sequence\_input)

average = GlobalAveragePooling1D()(embedded\_sequences)

predictions = Dense(N\_CLASSES, activation='softmax')(average)

model = Model(sequence\_input, predictions)

model.compile(loss='categorical\_crossentropy',

              optimizer='adam', metrics=['acc'])

model.fit(x\_train, y\_train, validation\_split=0.1,epochs=10, batch\_size=128)

output\_test = model.predict(x\_test)

print("ACCURACY:", roc\_auc\_score(y\_test,output\_test[:,1]))