import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

test = pd.read\_csv('drive/MyDrive/Colab Notebooks/Sentiment Analysis/test\_tweets.csv')

train = pd.read\_csv('drive/MyDrive/Colab Notebooks/Sentiment Analysis/train\_tweet.csv')

print(train.shape)

print(test.shape)

train.head()

test.head()

train.isnull().any()

test.isnull().any()

train[train['label'] == 0].head(10)

# checking out the postive comments from the train set

train[train['label'] == 1].head(10)

train['label'].value\_counts().plot.bar(color = 'pink', figsize = (6, 4))

# checking out the postive comments from the train set

train[train['label'] == 1].head(10)

train['label'].value\_counts().plot.bar(color = 'pink', figsize = (6, 4))

# adding a column to represent the length of the tweet

train['len'] = train['tweet'].str.len()

test['len'] = test['tweet'].str.len()

train.head(10)

train.groupby('label').describe()

train.groupby('len').mean()['label'].plot.hist(color = 'black', figsize = (6, 4),)

plt.title('variation of length')

plt.xlabel('Length')

plt.show()

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(stop\_words = 'english')

words = cv.fit\_transform(train.tweet)

sum\_words = words.sum(axis=0)

words\_freq = [(word, sum\_words[0, i]) for word, i in cv.vocabulary\_.items()]

words\_freq = sorted(words\_freq, key = lambda x: x[1], reverse = True)

frequency = pd.DataFrame(words\_freq, columns=['word', 'freq'])

frequency.head(30).plot(x='word', y='freq', kind='bar', figsize=(15, 7), color = 'blue')

plt.title("Most Frequently Occuring Words - Top 30")

from wordcloud import WordCloud

wordcloud = WordCloud(background\_color = 'white', width = 1000, height = 1000).generate\_from\_frequencies(dict(words\_freq))

plt.figure(figsize=(10,8))

plt.imshow(wordcloud)

plt.title("WordCloud - Vocabulary from Reviews", fontsize = 22)

normal\_words =' '.join([text for text in train['tweet'][train['label'] == 0]])

wordcloud = WordCloud(width=800, height=500, random\_state = 0, max\_font\_size = 110).generate(normal\_words)

plt.figure(figsize=(10, 7))

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

plt.axis('off')

plt.title('The Neutral Words')

plt.show()

negative\_words =' '.join([text for text in train['tweet'][train['label'] == 1]])

wordcloud = WordCloud(background\_color = 'cyan', width=800, height=500, random\_state = 0, max\_font\_size = 110).generate(negative\_words)

plt.figure(figsize=(10, 7))

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

plt.axis('off')

plt.title('The Negative Words')

plt.show()

import nltk

import re

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Make sure 'train' DataFrame is defined before running this code

def hashtag\_extract(x):

    hashtags = []

    for i in x:

        ht = re.findall(r"#(\w+)", i)

        hashtags.append(ht)

    return hashtags

# extracting hashtags from non-racist/sexist tweets

HT\_regular = hashtag\_extract(train['tweet'][train['label'] == 0])

HT\_regular = sum(HT\_regular, [])

# extracting hashtags from racist/sexist tweets

HT\_negative = hashtag\_extract(train['tweet'][train['label'] == 1])

HT\_negative = sum(HT\_negative, [])

# Plotting for non-racist/sexist tweets

a = nltk.FreqDist(HT\_regular)

d = pd.DataFrame({'Hashtag': list(a.keys()), 'Count': list(a.values())})

d = d.nlargest(columns="Count", n=20)

plt.figure(figsize=(16, 5))

ax = sns.barplot(data=d, x="Hashtag", y="Count")

ax.set(ylabel='Count')

plt.show()

# Plotting for racist/sexist tweets

a = nltk.FreqDist(HT\_negative)

d = pd.DataFrame({'Hashtag': list(a.keys()), 'Count': list(a.values())})

d = d.nlargest(columns="Count", n=20)

plt.figure(figsize=(16, 5))

ax = sns.barplot(data=d, x="Hashtag", y="Count")

ax.set(ylabel='Count')

plt.show()

# tokenizing the words present in the training set

tokenized\_tweet = train['tweet'].apply(lambda x: x.split())

# importing gensim

import gensim

# creating a word to vector model

model\_w2v = gensim.models.Word2Vec(

            tokenized\_tweet,

            vector\_size=200, # desired no. of features/independent variables

            window=5, # context window size

            min\_count=2,

            sg = 1, # 1 for skip-gram model

            hs = 0,

            negative = 10, # for negative sampling

            workers= 2, # no.of cores

            seed = 34)

model\_w2v.train(tokenized\_tweet, total\_examples= len(train['tweet']), epochs=20)

model\_w2v.wv.most\_similar(positive = "dinner")

model\_w2v.wv.most\_similar(positive = "cancer")

model\_w2v.wv.most\_similar(positive = "apple")

model\_w2v.wv.most\_similar(negative = "hate")

from tqdm import tqdm

tqdm.pandas(desc="progress-bar")

from gensim.models.doc2vec import TaggedDocument

def add\_label(twt):

    output = []

    for i, s in zip(twt.index, twt):

        output.append(TaggedDocument(s, ["tweet\_" + str(i)]))

    return output

# label all the tweets

labeled\_tweets = add\_label(tokenized\_tweet)

labeled\_tweets[:6]

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

nltk.download('stopwords')

train\_corpus = []

for i in range(0, 31962):

    review = re.sub('[^a-zA-Z]', ' ', train['tweet'][i])

    review = review.lower()

    review = review.split()

    ps = PorterStemmer()

    # stemming

    review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

    # joining them back with space

    review = ' '.join(review)

    train\_corpus.append(review)

test\_corpus = []

for i in range(0, 17197):

    review = re.sub('[^a-zA-Z]', ' ', test['tweet'][i])

    review = review.lower()

    review = review.split()

    ps = PorterStemmer()

    # stemming

    review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

    # joining them back with space

    review = ' '.join(review)

    test\_corpus.append(review)

# creating bag of words

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features=2500)

x = cv.fit\_transform(train\_corpus)

y = train.iloc[:, 1]

print(x.shape)

print(y.shape)

# creating bag of words

x\_test = cv.transform(test\_corpus)

print(x\_test.shape)

# splitting the training data into train and valid sets

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

x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(x, y, test\_size=0.25, random\_state=42)

# Convert sparse matrix to dense array

x\_train = x\_train.toarray()

x\_valid = x\_valid.toarray()

x\_test = x\_test.toarray()

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import f1\_score

# Standardization

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_valid = sc.transform(x\_valid)

x\_test = sc.transform(x\_test)

# Model training

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

# Predictions on the validation set

y\_pred = model.predict(x\_valid)

# Evaluation

print("Training Accuracy:", model.score(x\_train, y\_train))

print("Validation Accuracy:", model.score(x\_valid, y\_valid))

# calculating the f1 score for the validation set

print("f1 score :", f1\_score(y\_valid, y\_pred))

# confusion matrix

cm = confusion\_matrix(y\_valid, y\_pred)

print(cm)

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_valid)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Validation Accuracy :", model.score(x\_valid, y\_valid))

# calculating the f1 score for the validation set

print("f1 score :", f1\_score(y\_valid, y\_pred))

# confusion matrix

cm = confusion\_matrix(y\_valid, y\_pred)

print(cm)

# calculating the f1 score for the validation set

print("f1 score :", f1\_score(y\_valid, y\_pred))

# confusion matrix

cm = confusion\_matrix(y\_valid, y\_pred)

print(cm)

from sklearn.svm import SVC

model = SVC()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_valid)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Validation Accuracy :", model.score(x\_valid, y\_valid))

from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_valid)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Validation Accuracy :", model.score(x\_valid, y\_valid))

# calculating the f1 score for the validation set

print("f1 score :", f1\_score(y\_valid, y\_pred))

# confusion matrix

cm = confusion\_matrix(y\_valid, y\_pred)

print(cm)