!pip install kaggle

# API to fetch dataset from kaggle

!kaggle datasets download -d kazanova/sentiment140

from zipfile import ZipFile

dataset = '/content/sentiment140.zip'

with ZipFile(dataset,'r') as zip:

  zip.extractall()

  print('The dataset is extracted')

# importing dependencies

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import string

import nltk

import warnings

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

import nltk

nltk.download('stopwords')

nltk.download('punkt')

# printing the stop words in english

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

# data processing

twitter\_data = pd.read\_csv('/content/training.1600000.processed.noemoticon.csv',encoding='latin-1')

twitter\_data.head()

twitter\_data.shape

twitter\_data.columns

twitter\_data.head(5)

# naming the columns and read the data set again

columns\_names = ['target','id','date','flag','user','text']

twitter\_data = pd.read\_csv('/content/training.1600000.processed.noemoticon.csv',names=columns\_names,encoding='latin-1')

twitter\_data.shape

twitter\_data.head()

twitter\_data.isnull().sum()

# checking the distribution of target column

twitter\_data['target'].value\_counts()

# convertthe target '4' to '1'

twitter\_data['target'] = twitter\_data['target'].replace(4,1)

# 0== negative tweet

# 1== positive tweet

# Stemming(reducing word to its root word)

# example actor,acting = act

port\_stem = PorterStemmer()

def Stemming(content):

  stemmed\_content = re.sub('[^a-zA-Z]',' ',content)

  stemmed\_content = stemmed\_content.lower()

  stemmed\_content = stemmed\_content.split()

  stemmed\_content = [port\_stem.stem(word) for word in stemmed\_content if not word in stop\_words]

  stemmed\_content = ' '.join(stemmed\_content)

  return stemmed\_content

twitter\_data['stemmed\_text'] = twitter\_data['text'].apply(Stemming)

twitter\_data.head()

# Correcting the printing issue

print(twitter\_data[['text', 'stemmed\_text', 'target']].head())

print(twitter\_data['target'])

#  separating the data and label

X = twitter\_data['stemmed\_text'].values

Y = twitter\_data['target'].values

print(X)

print(Y)

# splitting the training data and test data

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, stratify=Y, random\_state=2)

print(X.shape, X\_train.shape, X\_test.shape)

print(X\_train)

print(X\_test)

# converting the texting data to numerical data

vectorization = TfidfVectorizer()

vectorization.fit(X\_train)

X\_train = vectorization.transform(X\_train)

X\_test = vectorization.transform(X\_test)

print(X\_test)

print(X\_train)

# Trainijng the machine learning model

# Logistic regression

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, Y\_train)

# Model Evaluation

# Accuracy score

# accuracy on training data

X\_train\_prediction = model.predict(X\_train)

training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)

print(  'Accuracy score on training data : ', training\_data\_accuracy)

# accuracy on test data

X\_test\_prediction = model.predict(X\_test)

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

print(  'Accuracy score on training data : ', test\_data\_accuracy)

# SAving the training model

import pickle

pickle.dump(model,

            open('model.pkl','wb'))

# using the save model for future predictions

# loading the save model

loaded\_model = pickle.load(open('model.pkl','rb'))

X\_new = X\_test[3]

print(Y\_test[3])

prediction = loaded\_model.predict(X\_new)

print(prediction)

if (prediction[0]==0):

  print('Negative Tweet')

else:

  print('Positive Tweet')

# Plotting the distribution of the target classes

plt.figure(figsize=(8, 5))

sns.countplot(x='target', data=twitter\_data)

plt.title('Distribution of Sentiment Classes')

plt.xlabel('Sentiment (0: Negative, 1: Positive)')

plt.ylabel('Count')

plt.xticks([0, 1], ['Negative', 'Positive'])

plt.show()

from sklearn.metrics import ConfusionMatrixDisplay

# Confusion matrix

cm = confusion\_matrix(Y\_test, X\_test\_prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['Negative', 'Positive'])

disp.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.show()

from sklearn.metrics import classification\_report

# Generate classification report

report = classification\_report(Y\_test, X\_test\_prediction, output\_dict=True)

df\_report = pd.DataFrame(report).transpose()

# Plotting performance metrics

df\_report[['precision', 'recall', 'f1-score']].iloc[:-1].plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics')

plt.ylabel('Score')

plt.xticks(rotation=45)

plt.show()

from wordcloud import WordCloud

# Generate a word cloud for positive tweets

positive\_tweets = twitter\_data[twitter\_data['target'] == 1]['text']

positive\_wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(' '.join(positive\_tweets))

plt.figure(figsize=(10, 5))

plt.imshow(positive\_wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud of Positive Tweets')

plt.show()

# Generate a word cloud for negative tweets

negative\_tweets = twitter\_data[twitter\_data['target'] == 0]['text']

negative\_wordcloud = WordCloud(width=800, height=400, background\_color='black').generate(' '.join(negative\_tweets))

plt.figure(figsize=(10, 5))

plt.imshow(negative\_wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud of Negative Tweets')

plt.show()







