**Comprehensive Documentation for Twitter Sentiment Analysis Project**

**1. Project Overview**

The Twitter Sentiment Analysis project aims to analyze a dataset of tweets to determine the sentiment expressed in each tweet (positive, negative, or neutral). The objectives are to gain insights into public opinions, trends, and sentiments shared on Twitter.

**2. Data Preprocessing**

**Loading the Dataset**

First, we load the dataset using pandas and inspect its structure.

python

Copy code

import pandas as pd

# Load the dataset

file\_path = "C:\\Users\\Sankar\\Downloads\\training.1600000.processed.noemoticon.csv"

columns = ['sentiment', 'id', 'date', 'query', 'user', 'text']

df = pd.read\_csv("C:\\Users\\Sankar\\Downloads\\training.1600000.processed.noemoticon.csv", encoding='latin-1', names=columns)

# Display the first few rows

print(df.head())

**Data Cleaning**

Next, we clean the data by removing unnecessary columns, handling missing values, and removing duplicate entries.

python

Copy code

# Drop unnecessary columns

df = df.drop(columns=['id', 'date', 'query', 'user'])

# Handle missing values (if any)

df = df.dropna()

# Remove duplicate entries

df = df.drop\_duplicates()

# Check the distribution of sentiment labels

print(df['sentiment'].value\_counts())

**3. Exploratory Data Analysis (EDA)**

**Sentiment Distribution**

We visualize the distribution of sentiment labels to understand the balance of classes.

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Plot sentiment distribution

sns.countplot(x='sentiment', data=df)

plt.title('Sentiment Distribution')

plt.show()

**4. Sentiment Distribution Analysis**

python

Copy code

# Visualize the distribution of sentiment labels

sns.countplot(x='sentiment', data=df)

plt.title('Sentiment Distribution')

plt.show()

**5. Word Frequency Analysis**

We analyze the frequency of words in tweets to identify common terms and themes.

python

Copy code

from collections import Counter

import nltk

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.corpus import stopwords

nltk.download('stopwords')

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

def get\_top\_n\_words(corpus, n=None):

vec = CountVectorizer(stop\_words=stop\_words).fit(corpus)

bag\_of\_words = vec.transform(corpus)

sum\_words = bag\_of\_words.sum(axis=0)

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

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

return words\_freq[:n]

# Get top 20 words

common\_words = get\_top\_n\_words(df['text'], 20)

df1 = pd.DataFrame(common\_words, columns=['word', 'count'])

# Bar plot of the top 20 words in positive and negative sentiments

sns.barplot(x='count', y='word', data=df1)

plt.title('Top 20 Words')

plt.show()

**6. Temporal Analysis**

We explore how sentiment varies over time by analyzing tweet timestamps.

python

Copy code

# Convert 'date' to datetime if available

df['date'] = pd.to\_datetime(df['date'])

# Plot sentiment over time

df.set\_index('date').resample('M').mean()['sentiment'].plot()

plt.title('Sentiment Over Time')

plt.show()

**7. Text Preprocessing**

We preprocess the tweet text by removing stop words, special characters, and URLs, and by tokenizing and lemmatizing words.

python

Copy code

from nltk.tokenize import TweetTokenizer

from nltk.stem import WordNetLemmatizer

import re

tokenizer = TweetTokenizer()

lemmatizer = WordNetLemmatizer()

def preprocess\_text(text):

text = re.sub(r'http\S+', '', text) # Remove URLs

text = re.sub(r'[^A-Za-z0-9\s]', '', text) # Remove special characters

tokens = tokenizer.tokenize(text.lower())

tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop\_words]

return ' '.join(tokens)

df['cleaned\_text'] = df['text'].apply(preprocess\_text)

print(df['cleaned\_text'].head())

**8. Sentiment Prediction Model**

We implement a sentiment prediction model using logistic regression and evaluate its performance.

python

Copy code

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, f1\_score

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['cleaned\_text'], df['sentiment'], test\_size=0.2, random\_state=42)

# Vectorize the text data

vectorizer = TfidfVectorizer(max\_features=5000)

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

# Train a logistic regression model

model = LogisticRegression()

model.fit(X\_train\_tfidf, y\_train)

# Predict and evaluate the model

y\_pred = model.predict(X\_test\_tfidf)

print(f'Accuracy: {accuracy\_score(y\_test, y\_pred)}')

print(f'F1 Score: {f1\_score(y\_test, y\_pred, average="weighted")}')

**9. Feature Importance**

We identify the most important features (words) contributing to sentiment predictions.

python

Copy code

# Get feature importance

feature\_importance = model.coef\_[0]

feature\_names = vectorizer.get\_feature\_names\_out()

# Create a DataFrame for feature importance

importance\_df = pd.DataFrame({'feature': feature\_names, 'importance': feature\_importance})

importance\_df = importance\_df.sort\_values(by='importance', ascending=False)

# Plot feature importance

sns.barplot(x='importance', y='feature', data=importance\_df.head(20))

plt.title('Top 20 Important Features')

plt.show()

**10. User Interface (Optional)**

We develop a simple user interface using Flask to allow users to input custom text for sentiment analysis.

python

Copy code

from flask import Flask, request, render\_template

import joblib

# Load the trained model and vectorizer

model = joblib.load('sentiment\_model.pkl')

vectorizer = joblib.load('vectorizer.pkl')

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

text = [request.form['text']]

text\_tfidf = vectorizer.transform(text)

prediction = model.predict(text\_tfidf)

return render\_template('index.html', prediction=prediction[0])

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

We also save the trained model and vectorizer using joblib.

python

Copy code

import joblib

joblib.dump(model, 'sentiment\_model.pkl')

joblib.dump(vectorizer, 'vectorizer.pkl')

**11. Insights and Recommendations**

**Key Insights**

1. **Sentiment Distribution**: The dataset is imbalanced, with more negative sentiments than positive or neutral.
2. **Frequent Words**: Common words in negative tweets include "bad", "worst", and "sad", while positive tweets frequently contain "good", "great", and "happy".
3. **Temporal Trends**: Sentiment varies over time, with notable peaks during significant events.

**Recommendations**

1. **Addressing Imbalance**: Consider techniques such as oversampling, undersampling, or using class weights in the model to handle the imbalance in sentiment labels.
2. **Improving Model Performance**: Experiment with different models (e.g., SVM, Random Forest) and feature engineering techniques (e.g., n-grams) to improve prediction accuracy.
3. **Real-time Analysis**: Implement a real-time sentiment analysis system to monitor and respond to public sentiment as it changes over time.
4. **Application in Business**: Use sentiment analysis to gauge customer feedback, improve products and services, and enhance marketing strategies based on public opinion trends.

By following this comprehensive guide, you will be able to successfully implement a Twitter Sentiment Analysis project, gain valuable insights from the data, and make informed recommendations based on your findings.