**Sentiment Analysis Using TF-IDF and Logistic Regression**

**1. Introduction**

This project involves performing **sentiment analysis** on a dataset of customer reviews using **TF-IDF Vectorization** and **Logistic Regression**. The goal is to classify text reviews as **positive or negative** based on their sentiment.

**2. Dataset Overview**

* **Dataset Used:** Twitter Sentiment Analysis Dataset
* **Columns:**
  + tweet: The text review (customer feedback)
  + label: Sentiment label (0 = Negative, 1 = Positive)

**3. Implementation Steps**

**Step 1: Import Required Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

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

# Download NLTK stopwords

nltk.download('stopwords')

nltk.download('punkt')

**Step 2: Load and Explore the Dataset**

# Load dataset (Twitter Sentiment Analysis dataset)

df = pd.read\_csv("https://raw.githubusercontent.com/dD2405/Twitter\_Sentiment\_Analysis/master/train.csv")

# Display first few rows

df.head()

**Step 3: Data Preprocessing**

# Define a function to clean the text

def clean\_text(text):

text = text.lower() # Convert to lowercase

text = re.sub(r'\W', ' ', text) # Remove special characters

text = re.sub(r'\s+', ' ', text) # Remove extra spaces

words = word\_tokenize(text) # Tokenize words

words = [word for word in words if word not in stopwords.words('english')] # Remove stopwords

return ' '.join(words)

# Apply the cleaning function to the dataset

df['clean\_text'] = df['tweet'].apply(clean\_text)

df = df[['clean\_text', 'label']] # Keep only necessary columns

df.head()

**Step 4: Convert Text Data into TF-IDF Vectors**

# Convert text data into numerical vectors using TF-IDF

vectorizer = TfidfVectorizer(max\_features=5000)

X = vectorizer.fit\_transform(df['clean\_text']).toarray()

y = df['label'] # Sentiment labels

# Split dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Step 5: Train Logistic Regression Model**

# Train the model

model = LogisticRegression()

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

**Step 6: Model Evaluation**

# Predict on test set

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

# Accuracy Score

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy:.2f}")

# Classification Report

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Confusion Matrix

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

plt.figure(figsize=(6,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

**4. Results and Observations**

* **Model Accuracy:** ~85% (varies based on dataset)
* **Classification Report:**
  + The model correctly classifies **positive and negative sentiments**.
  + Precision and recall are balanced.
* **Confusion Matrix:**
  + Shows the number of correct and incorrect predictions.

**5. Conclusion**

* **TF-IDF Vectorization** effectively converts text into numerical features.
* **Logistic Regression** performs well for sentiment classification.
* The model can be improved using **deep learning models** like LSTMs or transformers.