**Sentiment Analysis**

**Introduction:**

This project aims to perform sentiment analysis on a dataset using machine learning techniques. Sentiment analysis involves determining the sentiment or emotional tone expressed in a piece of text, often categorized as positive, negative, or neutral. The dataset used in this project is sourced from Kaggle and includes text content along with corresponding sentiment labels. This document provides a comprehensive overview of the sentiment analysis project, covering data preprocessing steps, model development, and evaluation results. The analysis aims to classify text sentiments using a machine learning model.

**Dataset**

The dataset used for sentiment analysis is sourced from Kaggle and contains text data labeled with sentiments (positive, negative, or neutral).

- Dataset Source: [Sentiment Analysis Dataset](https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset/data)

**## Data Exploration**

import pandas as pd

# Load the dataset

df = pd.read\_csv("Sentiment Analysis Dataset.csv", encoding='latin1')

# Explore dataset structure, features, and size

print(df.info())

print(df.head())

The dataset consists of columns such as 'SentimentText' (text content) and 'Sentiment' (sentiment labels).

**Data Preprocessing**

import re

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.naive\_bayes import MultinomialNB

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

# Text preprocessing functions

def preprocess\_text(text):

text = text.lower()

text = re.sub(r'[^a-zA-Z\s]', '', text)

text = ' '.join([WordNetLemmatizer().lemmatize(word) for word in text.split() if word not in stopwords.words('english')])

return text

# Apply preprocessing to the text content

df['ProcessedText'] = df['SentimentText'].apply(preprocess\_text)

The preprocessing steps include converting text to lowercase, removing special characters, and lemmatizing words.

**Exploratory Data Analysis (EDA)**

# Distribution of sentiment labels

sentiment\_distribution = df['Sentiment'].value\_counts()

sentiment\_distribution.plot(kind='bar', title='Sentiment Label Distribution')

plt.show()

This visualization provides insights into the distribution of sentiment labels, aiding in understanding the balance of sentiment classes.

**Text Vectorization**

# Text vectorization using TF-IDF

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)

X = tfidf\_vectorizer.fit\_transform(df['ProcessedText'])

The text data is converted into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency).

**Model Selection**

# Split the data into training and testing sets

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

# Naive Bayes model

nb\_model = MultinomialNB()

nb\_model.fit(X\_train, y\_train)

# Predictions

y\_pred = nb\_model.predict(X\_test)

# Evaluate the model

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

print(f'Accuracy: {accuracy}')

print(classification\_report(y\_test, y\_pred))

A Naive Bayes model is implemented for text classification, and its performance is evaluated using metrics like accuracy, precision, recall, and F1 score.

**Cross-Validation**

# Cross-validation

cv\_scores = cross\_val\_score(nb\_model, X, df['Sentiment'], cv=5)

print(f'Cross-validation Scores: {cv\_scores}')

print(f'Mean CV Score: {cv\_scores.mean()}')

Cross-validation techniques are applied to assess the generalization performance of the model and prevent overfitting.

**Model Interpretability**

# Feature importance (for Naive Bayes)

feature\_names = tfidf\_vectorizer.get\_feature\_names()

feature\_importance = nb\_model.coef\_[0]

# Display top N important features

top\_features = sorted(zip(feature\_importance, feature\_names), reverse=True)[:10]

print(f'Top 10 important features: {top\_features}')

Feature importance is analyzed to understand which words contribute most to sentiment predictions.

**Evaluation Metrics**

# Confusion matrix, precision-recall curves, ROC-AUC (optional)

# Use sklearn.metrics functions for these evaluations

Evaluation metrics such as confusion matrix, precision-recall curves, and ROC-AUC can be employed to assess the model's performance.

**Conclusion**

The sentiment analysis project involves preprocessing text data, building a Naive Bayes model, and evaluating its performance. Insights gained from exploratory data analysis and model interpretability contribute to a better understanding of the sentiment analysis process.