

```

# =====
# Project Name : NLP-practice-project
#
# Dataset Name : IMDB Dataset.csv
# Dataset Source : Kaggle
# Dataset Link :
# https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
#
# Dataset Information:
# - Total Samples      : 50,000 movie reviews
# - Classes            : 2 (Positive, Negative)
# - Class Distribution  : Balanced (25,000 positive / 25,000 negative)
# - Text Length        : Variable-length movie reviews
# - Task Type          : Binary Text Classification (Sentiment Analysis)
#
# =====

# -----
# 1. Install Required Libraries
# -----
!pip install kaggle nltk scikit-learn pandas matplotlib seaborn joblib

# -----
# 2. Configure Kaggle API
# -----
# Note:
# Kaggle API is used to download the dataset programmatically
# Upload kaggle.json (API token) before running this cell

import os
os.makedirs("/root/.kaggle", exist_ok=True)
!cp kaggle.json /root/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json

# -----
# 3. Download Dataset
# -----
# This command downloads the IMDB Dataset and extracts
# the file "IMDB Dataset.csv" into the working directory

!kaggle datasets download -d lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
!unzip imdb-dataset-of-50k-movie-reviews.zip

# -----
# 4. Import Libraries
# -----
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns

```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

nltk.download('stopwords')
nltk.download('wordnet')

# -----
# 5. Load Dataset
# -----
# The dataset contains two main columns:
# - review    : raw text data (movie reviews)
# - sentiment : target label (positive / negative)

df = pd.read_csv("IMDB Dataset.csv")

print("Dataset Shape:", df.shape)
display(df.head())

# -----
# 6. Dataset Label Distribution (Visualization)
# -----
# This visualization confirms that the dataset is balanced,
# which helps reduce bias during model training.

plt.figure(figsize=(6,4))
sns.countplot(x='sentiment', data=df)
plt.title("Sentiment Label Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.show()

# -----
# 7. Text Preprocessing & Feature Engineering
# -----
# Feature Engineering Steps Applied:
# 1. Convert text to lowercase
# 2. Remove punctuation and special characters
# 3. Tokenize text into words
# 4. Remove stopwords (common non-informative words)
# 5. Apply lemmatization (convert words to base form)
#
# These steps improve model performance by reducing noise
# and standardizing the textual features.

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = text.lower()
```

```

text = re.sub(r'^a-z\s', '', text)
words = text.split()
words = [
    lemmatizer.lemmatize(word)
    for word in words
    if word not in stop_words
]
return " ".join(words)

df['clean_review'] = df['review'].apply(preprocess_text)

# Show example before and after preprocessing
print("\nOriginal Review Sample:")
print(df['review'][0][:300])

print("\nCleaned Review Sample:")
print(df['clean_review'][0][:300])

# Encode sentiment labels into numeric form
# Positive → 1, Negative → 0
df['label'] = df['sentiment'].map({'positive': 1, 'negative': 0})

# -----
# 8. Train-Test Split
# -----
# The dataset is split into:
# - 80% Training data
# - 20% Testing data
#
# Stratification ensures equal class distribution
# in both training and testing sets.

X = df['clean_review']
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Training Samples:", X_train.shape[0])
print("Testing Samples :", X_test.shape[0])

# -----
# 9. Feature Representation (TF-IDF)
# -----
# TF-IDF (Term Frequency-Inverse Document Frequency)
# converts text into numerical vectors by measuring
# word importance across documents.
#
# max_features=5000 limits the feature space to the
# top 5000 most informative terms.

tfidf = TfidfVectorizer(max_features=5000)

```

```

X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print("TF-IDF Feature Matrix Shape:", X_train_tfidf.shape)

# -----
# 10. Machine Learning Models Used
# -----
# Model 1: Logistic Regression
# - A linear classification model
# - Uses the sigmoid activation function internally
#   to map outputs to probabilities (0-1)
#
# Model 2: Multinomial Naive Bayes
# - Probabilistic classifier based on Bayes' theorem
# - Well-suited for text classification problems
#
# Note:
# Activation functions are implicit in ML models.
# Explicit activation functions (ReLU, Softmax)
# are typically used in Deep Learning models.

lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_tfidf, y_train)

nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)

# -----
# 11. Model Evaluation & Output Visualization
# -----
# Evaluation Metrics Used:
# - Accuracy
# - Precision
# - Recall
# - F1-score
# - Confusion Matrix (visual)

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)

    print(f"\n{model_name} Evaluation Results")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4,4))
    sns.heatmap(
        cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=['Negative', 'Positive'],
        yticklabels=['Negative', 'Positive']
    )
    plt.title(f"{model_name} - Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

```

```
evaluate_model(lr_model, X_test_tfidf, y_test, "Logistic Regression")
evaluate_model(nb_model, X_test_tfidf, y_test, "Naive Bayes")

# -----
# 12. Save Trained Models
# -----
# The trained model and TF-IDF vectorizer are saved
# for future inference or deployment.

import joblib
joblib.dump(lr_model, "logistic_regression_sentiment_model.pkl")
joblib.dump(tfidf, "tfidf_vectorizer.pkl")

print("\n NLP-practice-project completed successfully")
```

