**Python Project: Sentiment Analysis**

**Basic Information and Project Components**

* **Project:** Sentiment Analysis
* **Objective:** To analyze and classify the sentiment of text data from movie reviews.
* **Tools and Techniques:**
  + **Libraries:** pandas, numpy, matplotlib, seaborn, spacy, transformers, torch, sklearn
  + **Machine Learning Models:** Naive Bayes, Logistic Regression, Ensemble Model (Naive Bayes & Logistic Regression), Hugging Face Transformer Model
  + **Data Source:** https://huggingface.co/datasets/cornell-movie-review-data/rotten\_tomatoes

**Steps**

1. **Data Preparation:**
   * Loaded and cleaned the dataset.
   * Applied text preprocessing (lowercasing, removing special characters, tokenization, lemmatization).
   * Used spacy for text cleaning and tokenization.
2. **Feature Extraction:**
   * Used BERT model to extract embeddings for the text data.
   * Tokenized text and generated embeddings with BertTokenizer and BertModel.
3. **Model Training:**
   * Split data into training and testing sets.
   * Trained Naive Bayes and Logistic Regression models.
   * Created an ensemble model combining both classifiers.
   * Applied Hugging Face Transformer for sentiment prediction.
4. **Evaluation:**
   * Predicted sentiments on the test set.
   * Calculated accuracy, precision, recall, and F1 score for each model.
   * Visualized model performance using bar plots.

**Results and Insights**

* **Model Performance:**
  + **Naive Bayes:**
    - Accuracy: 73.36%
    - Precision: 73.68%
    - Recall: 73.36%
    - F1 Score: 73.39%
  + **Logistic Regression:**
    - Accuracy: 68.22%
    - Precision: 68.58%
    - Recall: 68.22%
    - F1 Score: 68.25%
  + **Ensemble Model:**
    - Accuracy: 73.83%
    - Precision: 74.09%
    - Recall: 73.83%
    - F1 Score: 73.86%
  + **Hugging Face Transformer Model:**
    - Accuracy: 78.71%
    - Precision: 78.73%
    - Recall: 78.71%
    - F1 Score: 78.70%
* **Business Insights:**
  + **Customer Feedback Analysis:** Helps in understanding customer sentiment towards products or services, enabling better decision-making.
  + **Market Research:** Analyzes large volumes of reviews to gauge public opinion on movies, products, or brands.
  + **Brand Monitoring:** Tracks and manages the company's reputation by identifying positive and negative sentiments in real-time.
  + **Improved Customer Experience:** Allows companies to respond to negative feedback promptly and improve their offerings based on sentiment trends.

This sentiment analysis project showcases the ability to preprocess text data, extract meaningful features using BERT, and apply various machine learning models, including advanced Hugging Face transformers, to derive insights from textual data. These insights can significantly benefit various business applications.

import subprocess  
import re  
import spacy  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from datasets import load\_dataset  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import VotingClassifier  
from transformers import pipeline, BertTokenizer, BertModel  
import torch  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
  
# Ensure the spacy model is downloaded  
subprocess.run(["python", "-m", "spacy", "download", "en\_core\_web\_sm"])  
  
# Load the spacy model  
spacy\_en = spacy.load('en\_core\_web\_sm')  
stopwords = spacy\_en.Defaults.stop\_words  
  
def clean\_text(text):  
 text = text.lower()  
 text = re.sub(r"[.!?/\-\_\*:\",'()#@$%^&]", " ", text)  
 text = re.sub(r"\s+", " ", text).strip()  
 return text  
  
def tokenize\_text(text):  
 tokens = spacy\_en(text)  
 result = []  
 for token in tokens:  
 lemma = token.lemma\_  
 if lemma not in stopwords:  
 result.append(lemma)  
 return ' '.join(result)  
  
# Load the dataset  
dataset = load\_dataset('cornell-movie-review-data/rotten\_tomatoes')  
df = pd.DataFrame(dataset['test'])  
  
print(df.count())  
  
df["text"] = df["text"].apply(clean\_text)  
df["text"] = df["text"].apply(tokenize\_text)  
  
print(df.head())  
  
print(f"Distribution of classes in the dataset: {df['label'].value\_counts()}")  
  
tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')  
model = BertModel.from\_pretrained('bert-base-uncased')  
  
def get\_bert\_embeddings(text):  
 inputs = tokenizer(text, return\_tensors='pt', truncation=True, padding=True, max\_length=512)  
 with torch.no\_grad():  
 outputs = model(\*\*inputs)  
 return outputs.last\_hidden\_state.mean(dim=1).squeeze().numpy()  
  
df['bert\_embeddings'] = df['text'].apply(get\_bert\_embeddings)  
X\_bert = np.vstack(df['bert\_embeddings'])  
y\_bert = df['label'].values  
  
X\_train\_bert, X\_test\_bert, y\_train\_bert, y\_test\_bert = train\_test\_split(X\_bert, y\_bert, test\_size=0.2, random\_state=42)  
  
gnb = GaussianNB()  
logreg = LogisticRegression(max\_iter=200)  
gnb.fit(X\_train\_bert, y\_train\_bert)  
logreg.fit(X\_train\_bert, y\_train\_bert)  
  
y\_pred\_gnb = gnb.predict(X\_test\_bert)  
y\_pred\_logreg = logreg.predict(X\_test\_bert)  
  
accuracy\_gnb = accuracy\_score(y\_test\_bert, y\_pred\_gnb) \* 100  
precision\_gnb = precision\_score(y\_test\_bert, y\_pred\_gnb, average='weighted') \* 100  
recall\_gnb = recall\_score(y\_test\_bert, y\_pred\_gnb, average='weighted') \* 100  
f1\_gnb = f1\_score(y\_test\_bert, y\_pred\_gnb, average='weighted') \* 100  
  
accuracy\_logreg = accuracy\_score(y\_test\_bert, y\_pred\_logreg) \* 100  
precision\_logreg = precision\_score(y\_test\_bert, y\_pred\_logreg, average='weighted') \* 100  
recall\_logreg = recall\_score(y\_test\_bert, y\_pred\_logreg, average='weighted') \* 100  
f1\_logreg = f1\_score(y\_test\_bert, y\_pred\_logreg, average='weighted') \* 100  
  
print(f"Naive Bayes Model Performance:")  
print(f"Accuracy: {accuracy\_gnb:.2f}%")  
print(f"Precision: {precision\_gnb:.2f}%")  
print(f"Recall: {recall\_gnb:.2f}%")  
print(f"F1 Score: {f1\_gnb:.2f}%")  
  
print(f"\nLogistic Regression Model Performance:")  
print(f"Accuracy: {accuracy\_logreg:.2f}%")  
print(f"Precision: {precision\_logreg:.2f}%")  
print(f"Recall: {recall\_logreg:.2f}%")  
print(f"F1 Score: {f1\_logreg:.2f}%")  
  
ensemble\_model = VotingClassifier(estimators=[('gnb', gnb), ('logreg', logreg)], voting='soft')  
ensemble\_model.fit(X\_train\_bert, y\_train\_bert)  
  
y\_pred\_ensemble = ensemble\_model.predict(X\_test\_bert)  
accuracy\_ensemble = accuracy\_score(y\_test\_bert, y\_pred\_ensemble) \* 100  
precision\_ensemble = precision\_score(y\_test\_bert, y\_pred\_ensemble, average='weighted') \* 100  
recall\_ensemble = recall\_score(y\_test\_bert, y\_pred\_ensemble, average='weighted') \* 100  
f1\_ensemble = f1\_score(y\_test\_bert, y\_pred\_ensemble, average='weighted') \* 100  
  
print(f"\nEnsemble Model Performance (Naive Bayes & Logistic Regression):")  
print(f"Accuracy: {accuracy\_ensemble:.2f}%")  
print(f"Precision: {precision\_ensemble:.2f}%")  
print(f"Recall: {recall\_ensemble:.2f}%")  
print(f"F1 Score: {f1\_ensemble:.2f}%")  
  
models = ['Naive Bayes', 'Logistic Regression', 'Ensemble']  
accuracies = [accuracy\_gnb / 100, accuracy\_logreg / 100, accuracy\_ensemble / 100]  
plt.figure(figsize=(10, 6))  
sns.barplot(x=models, y=accuracies, palette="viridis", hue=models, dodge=False)  
plt.title('Model Performance')  
plt.ylabel('Accuracy')  
plt.ylim(0, 1)  
plt.show()  
  
def analyze\_sentiment\_ensemble(text):  
 cleaned\_text = clean\_text(text)  
 tokenized\_text = tokenize\_text(cleaned\_text)  
 bert\_embedding = get\_bert\_embeddings(tokenized\_text).reshape(1, -1)  
 ensemble\_result = ensemble\_model.predict(bert\_embedding)[0]  
 ensemble\_sentiment = "Positive" if ensemble\_result == 1 else "Negative"  
 return ensemble\_sentiment  
  
def analyze\_sentiment\_hf(text):  
 sentiment\_pipeline = pipeline('sentiment-analysis')  
 result = sentiment\_pipeline(text)[0]  
 return result  
  
example\_text = "One of the best musicals I've ever seen!"  
print(f"\nText: {example\_text}")  
  
ensemble\_sentiment = analyze\_sentiment\_ensemble(example\_text)  
print(f"Predicted sentiment (Ensemble): {ensemble\_sentiment}")  
  
hf\_result = analyze\_sentiment\_hf(example\_text)  
print(f"Sentiment (Hugging Face): {hf\_result['label']}")  
print(f"Hugging Face prediction details: {hf\_result}")  
  
df["predicted\_sentiment"] = ensemble\_model.predict(np.vstack(df['bert\_embeddings']))  
plt.figure(figsize=(12, 6))  
sns.countplot(x='predicted\_sentiment', data=df, palette="viridis", hue='predicted\_sentiment', legend=False)  
plt.title('Distribution of Predicted Sentiment in Test Dataset')  
plt.xlabel('Sentiment')  
plt.ylabel('Number of Samples')  
plt.xticks(ticks=[0, 1], labels=['Positive', 'Negative'])  
plt.show()  
  
df["hf\_sentiment"] = df["text"].apply(lambda x: analyze\_sentiment\_hf(x)['label'])  
df["hf\_sentiment"] = df["hf\_sentiment"].map({'POSITIVE': 1, 'NEGATIVE': 0})  
  
comparison\_accuracy = accuracy\_score(df["label"], df["hf\_sentiment"]) \* 100  
comparison\_precision = precision\_score(df["label"], df["hf\_sentiment"], average='weighted') \* 100  
comparison\_recall = recall\_score(df["label"], df["hf\_sentiment"], average='weighted') \* 100  
comparison\_f1 = f1\_score(df["label"], df["hf\_sentiment"], average='weighted') \* 100  
  
print(f"\nHugging Face Model Comparison")  
print(f"Accuracy: {comparison\_accuracy:.2f}%")  
print(f"Precision: {comparison\_precision:.2f}%")  
print(f"Recall: {comparison\_recall:.2f}%")  
print(f"F1 Score: {comparison\_f1:.2f}%")