Movie Genre Classification using NLP and Machine Learning (TF-IDF + LR, NB, SVM)

Project Title: Movie Genre Classification using Natural Language Processing (NLP) and Machine Learning

Objective: To accurately classify movies into their respective genres based on their plot summaries. This project involves using NLP techniques to process text data and applying machine learning models to perform the classification.

```
#import the required libraries
import pandas as pd
import re
import os
import zipfile
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import LabelEncoder
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.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
```

Data Collection and Loading

Source: The project uses a dataset from Kaggle titled "Genre Classification Dataset IMDb".

Datasets: train_data.txt, test_data.txt, test_data_solution.txt

```
# --- Data Collection ---
def extract_dataset(zip_file_path, extracted_dir):
    if not os.path.exists(extracted_dir):
        with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
            zip_ref.extractall(extracted_dir)
        print(f"Dataset extracted to '{extracted_dir}'")
    else:
        print("Dataset already extracted.")

extracted_dir = 'Genre_Classification_Dataset'
train_data_path = os.path.join(extracted_dir, 'train_data.txt')
test_data_path = os.path.join(extracted_dir, 'test_data.txt')
solution_data_path = os.path.join(extracted_dir, 'test_data_solution.txt')

# Add the path to the zip file here
zip_file_path = '/content/sample_data/Genre Classification Dataset IMDb.zip'

# Extract the dataset
extract_dataset(zip_file_path, extracted_dir)
```

```
Dataset extracted to 'Genre_Classification_Dataset'
```

```
# Load the datasets
column_names_train = ['ID', 'Title', 'Genre', 'Plot']
```

```
df_train = pd.read_csv('/content/Genre_Classification_Dataset/Genre Classification Dataset/

column_names_test = ['ID', 'Title', 'Plot']
df_test = pd.read_csv('/content/Genre_Classification_Dataset/Genre Classification Dataset/t

column_names_solution = ['ID', 'Title', 'Genre', 'Plot']
df_solution = pd.read_csv('/content/Genre_Classification_Dataset/Genre Classification Datas

print("Training data loaded successfully.")
print(df_train.head())
print("\nTest data loaded successfully.")
print(df_test.head())
```

```
Training data loaded successfully.
   ID  ...                                               Plot
0   1  ...   Listening in to a conversation between his do...
1   2  ...   A brother and sister with a past incestuous r...
2   3  ...   As the bus empties the students for their fie...
3   4  ...   To help their unemployed father make ends mee...
4   5  ...   The film's title refers not only to the un-re...

[5 rows x 4 columns]

Test data loaded successfully.
   ID  ...                                               Plot
0   1  ...   L.R. Brane loves his life - his car, his apar...
1   2  ...   Spain, March 1964: Quico is a very naughty ch...
2   3  ...   One year in the life of Albin and his family ...
3   4  ...   His father has died, he hasn't spoken with hi...
4   5  ...   Before he was known internationally as a mart...

[5 rows x 3 columns]
```

Data Preprocessing Purpose: The raw movie plots need to be cleaned before they can be used by the models. The clean_text function performs the following steps:

Lowercasing: Converts all text to lowercase to ensure consistency.

Punctuation and Number Removal: Removes all punctuation and numerical digits.

Tokenization: Breaks the text into individual words (tokens).

Stop Word Removal: Eliminates common English words (e.g., "the," "a," "is") that don't add significant meaning.

Lemmatization: Reduces words to their base or root form (e.g., "running" becomes "run") to reduce vocabulary size and improve consistency.

```
# --- Preprocessing ---
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('punkt_tab')

def clean_text(text):
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub(r'\d+', '', text)
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)

    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_wo
```

```
    return ' '.join(lemmatized_tokens)

df_train['Cleaned_Plot'] = df_train['Plot'].apply(clean_text)
df_test['Cleaned_Plot'] = df_test['Plot'].apply(clean_text)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
# See all unique genres in the dataset
print(df_train['Genre'].unique())
print(df_train['Genre'].value_counts())
```

```
[' drama ' ' thriller ' ' adult ' ' documentary ' ' comedy ' ' crime '
 ' reality-tv ' ' horror ' ' sport ' ' animation ' ' action ' ' fantasy '
 ' short ' ' sci-fi ' ' music ' ' adventure ' ' talk-show ' ' western '
 ' family ' ' mystery ' ' history ' ' news ' ' biography ' ' romance '
 ' game-show ' ' musical ' ' war ']
Genre
drama           13613
documentary     13096
comedy           7447
short            5073
horror           2204
thriller         1591
action           1315
western          1032
reality-tv        884
family            784
adventure         775
music             731
romance           672
sci-fi            647
adult             590
crime             505
animation         498
sport             432
talk-show         391
fantasy           323
mystery           319
musical           277
biography         265
history           243
game-show         194
news              181
war               132
Name: count, dtype: int64
```

Feature Representation (TF-IDF) : Term Frequency-Inverse Document Frequency

Purpose: Machine learning models cannot directly process text. TF-IDF Vectorization is used to convert the cleaned text data into a numerical representation.

```
# --- Feature Representation (TF-IDF) ---

# Separate features (X) and target (y)
X_train_text = df_train['Cleaned_Plot']
y_train_genre = df_train['Genre']
X_test_text = df_test['Cleaned_Plot']

# Convert genres to numerical labels using LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_genre)
```

```
# Create a TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=15000, ngram_range=(1, 2))
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_text)
X_test_tfidf = tfidf_vectorizer.transform(X_test_text)

print("\nData transformed using TF-IDF.")
print(f"Shape of X_train_tfidf: {X_train_tfidf.shape}")
print(f"Shape of X_test_tfidf: {X_test_tfidf.shape}")
```

```
Data transformed using TF-IDF.
Shape of X_train_tfidf: (54214, 15000)
Shape of X_test_tfidf: (54200, 15000)
```

========================Model Training & Evaluation========================

Logistic Regression: A powerful and simple model that's great for text classification. It's often the first model to try because it's fast and reliable.

```
# --- Model 1: Logistic Regression ---

print("\n--- Training and Evaluating Logistic Regression ---")
lr_classifier = LogisticRegression(max_iter=1000, solver='liblinear', random_state=42)
lr_classifier.fit(X_train_tfidf, y_train_encoded)
y_pred_lr_encoded = lr_classifier.predict(X_test_tfidf)

# Decode the predicted labels back to genre names
y_pred_lr_genre = label_encoder.inverse_transform(y_pred_lr_encoded)
y_true_genre = df_solution['Genre']

# Evaluate the model
accuracy_lr = accuracy_score(y_true_genre, y_pred_lr_genre)
print(f"Accuracy for Logistic Regression: {accuracy_lr:.4f}")
print("Classification Report (Logistic Regression):")
print(classification_report(y_true_genre, y_pred_lr_genre, zero_division=0))
```

```
--- Training and Evaluating Logistic Regression ---
Accuracy for Logistic Regression: 0.5864
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

      action       0.52      0.26      0.35      1314
       adult       0.65      0.22      0.33       590
   adventure       0.70      0.15      0.25       775
   animation       0.53      0.04      0.08       498
   biography       0.00      0.00      0.00       264
      comedy       0.54      0.58      0.56      7446
       crime       0.45      0.03      0.05       505
 documentary       0.66      0.87      0.75     13096
       drama       0.54      0.80      0.64     13612
      family       0.56      0.08      0.14       783
     fantasy       0.75      0.03      0.05       322
   game-show       0.88      0.53      0.66       193
     history       0.00      0.00      0.00       243
      horror       0.67      0.57      0.62      2204
       music       0.71      0.39      0.50       731
     musical       0.29      0.01      0.01       276
     mystery       0.00      0.00      0.00       318
        news       0.82      0.05      0.09       181
  reality-tv       0.55      0.15      0.24       883
     romance       0.50      0.01      0.02       672
      sci-fi       0.62      0.23      0.33       646
       short       0.51      0.30      0.38      5072
```

```
       sport       0.74      0.20      0.32       431
   talk-show       0.69      0.16      0.25       391
    thriller       0.40      0.09      0.15      1590
         war       0.00      0.00      0.00       132
     western       0.92      0.73      0.81      1032

    accuracy                           0.59     54200
   macro avg       0.52      0.24      0.28     54200
weighted avg       0.57      0.59      0.54     54200
```

Naive Bayes: A fast and effective probabilistic model, particularly well-suited for text data with many features.

```
# --- Model 2: Naive Bayes ---

print("\n--- Training and Evaluating Naive Bayes ---")
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_tfidf, y_train_encoded)
y_pred_nb_encoded = nb_classifier.predict(X_test_tfidf)

# Decode the predicted labels back to genre names
y_pred_nb_genre = label_encoder.inverse_transform(y_pred_nb_encoded)

# Evaluate the model
accuracy_nb = accuracy_score(y_true_genre, y_pred_nb_genre)
print(f"Accuracy for Naive Bayes: {accuracy_nb:.4f}")
print("Classification Report (Naive Bayes):")
print(classification_report(y_true_genre, y_pred_nb_genre, zero_division=0))
```

```
--- Training and Evaluating Naive Bayes ---
Accuracy for Naive Bayes: 0.5120
Classification Report (Naive Bayes):
              precision    recall  f1-score   support

      action       0.62      0.03      0.06      1314
       adult       0.56      0.04      0.07       590
   adventure       0.80      0.06      0.11       775
   animation       0.00      0.00      0.00       498
   biography       0.00      0.00      0.00       264
      comedy       0.53      0.42      0.47      7446
       crime       0.00      0.00      0.00       505
 documentary       0.56      0.89      0.69     13096
       drama       0.45      0.84      0.59     13612
      family       0.00      0.00      0.00       783
     fantasy       0.00      0.00      0.00       322
   game-show       1.00      0.03      0.05       193
     history       0.00      0.00      0.00       243
      horror       0.77      0.24      0.36      2204
       music       1.00      0.02      0.03       731
     musical       0.00      0.00      0.00       276
     mystery       0.00      0.00      0.00       318
        news       0.00      0.00      0.00       181
  reality-tv       0.00      0.00      0.00       883
     romance       0.00      0.00      0.00       672
      sci-fi       0.00      0.00      0.00       646
       short       0.67      0.09      0.16      5072
       sport       0.91      0.05      0.09       431
   talk-show       0.00      0.00      0.00       391
    thriller       0.30      0.00      0.00      1590
         war       0.00      0.00      0.00       132
     western       0.99      0.42      0.59      1032

    accuracy                           0.51     54200
   macro avg       0.34      0.12      0.12     54200
```

```
weighted avg       0.50      0.51      0.42     54200
```

LinearSVC (Support Vector Machine): Efficient model for large-scale classification tasks.

```
# --- Model 3: Support Vector Machines (SVM) ---

print("\n--- Training and Evaluating LinearSVC ---")
# Use LinearSVC, which is optimized for large-scale linear classification
svm_classifier = LinearSVC(random_state=42, max_iter=2000)
svm_classifier.fit(X_train_tfidf, y_train_encoded)
y_pred_svm_encoded = svm_classifier.predict(X_test_tfidf)

# Decode the predicted labels back to genre names
y_pred_svm_genre = label_encoder.inverse_transform(y_pred_svm_encoded)
y_true_genre = df_solution['Genre']

# Evaluate the model
accuracy_svm = accuracy_score(y_true_genre, y_pred_svm_genre)
print(f"Accuracy for LinearSVC: {accuracy_svm:.4f}")
print("Classification Report (LinearSVC):")
print(classification_report(y_true_genre, y_pred_svm_genre, zero_division=0))
```

```
--- Training and Evaluating LinearSVC ---
Accuracy for LinearSVC: 0.5748
Classification Report (LinearSVC):
               precision    recall  f1-score   support

      action       0.41      0.33      0.37      1314
       adult       0.60      0.41      0.49       590
   adventure       0.46      0.22      0.30       775
   animation       0.33      0.14      0.20       498
   biography       0.11      0.01      0.01       264
      comedy       0.53      0.57      0.55      7446
       crime       0.21      0.07      0.11       505
 documentary       0.69      0.81      0.74     13096
       drama       0.57      0.70      0.63     13612
      family       0.31      0.14      0.19       783
     fantasy       0.27      0.08      0.12       322
   game-show       0.80      0.65      0.72       193
     history       0.11      0.01      0.02       243
      horror       0.61      0.62      0.61      2204
       music       0.60      0.50      0.54       731
     musical       0.28      0.07      0.11       276
     mystery       0.21      0.04      0.07       318
        news       0.56      0.13      0.21       181
  reality-tv       0.46      0.29      0.35       883
     romance       0.28      0.08      0.12       672
      sci-fi       0.49      0.35      0.41       646
       short       0.41      0.35      0.38      5072
       sport       0.56      0.40      0.46       431
   talk-show       0.51      0.28      0.36       391
    thriller       0.30      0.18      0.22      1590
         war       0.58      0.16      0.25       132
     western       0.83      0.83      0.83      1032

    accuracy                           0.57     54200
   macro avg       0.45      0.31      0.35     54200
weighted avg       0.55      0.57      0.55     54200
```

Knowing the BEST Accuracy Model.........

```
# To know the best model from three classifiers
classifiers = {
    'Logistic Regression': lr_classifier,
    'Naive Bayes': nb_classifier,
    'Support Vector Machines': svm_classifier
}
# Storing the accuracies of three classifiers
accuracies = {
    'Logistic Regression': accuracy_lr,
    'Naive Bayes': accuracy_nb,
    'Support Vector Machines': accuracy_svm
}

# Find the name of the classifier with the highest accuracy
best_model_name = max(accuracies, key=accuracies.get)
best_model = classifiers[best_model_name]

print(f"The best model is: {best_model_name} with an accuracy of {accuracies[best_model_nam
```

```
The best model is: Logistic Regression with an accuracy of 0.5864
```

==========================Final Prediction==========================

```
# Make predictions on the test dataset using the best model
final_predictions_encoded = best_model.predict(X_test_tfidf)
final_predictions_genre = label_encoder.inverse_transform(final_predictions_encoded)
df_test['Predicted_Genre'] = final_predictions_genre

# Creating the output CSV file
output_path = 'predicted_genre.csv'
df_test[['ID', 'Title', 'Predicted_Genre']].to_csv(output_path, index=False)

print(f"\nFinal predictions saved to '{output_path}'")
print("\nFirst 5 rows of the output file:")
print(df_test[['ID', 'Title', 'Predicted_Genre']].head())
```

```
Final predictions saved to 'predicted_genre.csv'

First 5 rows of the output file:
   ID                        Title Predicted_Genre
0   1          Edgar's Lunch (1998)           drama
1   2      La guerra de papá (1977)           drama
2   3   Off the Beaten Track (2010)     documentary
3   4         Meu Amigo Hindu (2015)           drama
4   5              Er nu zhai (1955)           drama
```

=================== simple use case ======================

```
# ==== Sample Prediction using Logistic Regression ====

# New movie plot summary to test
sample_plot = ["A disgraced detective is assigned to a high-stakes case involving a serial k
cleaned_sample_plot = [clean_text(plot) for plot in sample_plot] #preprocessing
sample_plot_tfidf = tfidf_vectorizer.transform(cleaned_sample_plot) #Transform using vectori

# Make a prediction using the trained Logistic Regression model
predicted_genre_encoded = lr_classifier.predict(sample_plot_tfidf)
predicted_genre = label_encoder.inverse_transform(predicted_genre_encoded)

print(f"\n--- Prediction for a Sample Movie Plot ---")
print(f"Sample Plot: '{sample_plot[0]}'")
```

```
print(f"Sample Plot: '{sample_plot[0]}'")
print(f"Predicted Genre: {predicted_genre[0]}")
```

```
--- Prediction for a Sample Movie Plot ---
Sample Plot: 'A disgraced detective is assigned to a high-stakes case involving a serial kil
Predicted Genre:  thriller
```