movie-genre-classifier

[2]:
```
import pandas as pd
import nltk
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from google.colab import drive
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

[3]:
```
True
```

[4]:
```
# Define column names for your DataFrame
head_column = ['serial_number','movie_title', 'genre', 'movie_description']

df = pd.read_csv(r'./drive/MyDrive/Colab Notebooks/train_data.txt', sep=':::',␣
 ↪names= head_column, engine='python')

df['year'] = df['movie_title'].str.extract(r'\((\d{4})\)', expand=False)
df['movie_title'] = df['movie_title'].str.extract(r'(.+?) \(\d+\)')
```

```
print(df.count())
```

```
serial_number        54214
movie_title          49867
genre                54214
movie_description    54214
year                 49867
dtype: int64
```

[5]:
```
#Data Preprocessing

#Remove stop words
stop_words = set(stopwords.words('english'))

# Function to remove stop words
def remove_stop_words(text):
    tokens = word_tokenize(text)
    filtered_tokens = [word for word in tokens if word.lower() not in␣
 ↪stop_words]
    return ' '.join(filtered_tokens)

# Apply stop words removal to the 'movie_description' column
df['movie_description'] = df['movie_description'].apply(remove_stop_words)

print(df.head())
```

```
   serial_number                movie_title       genre  \
0              1      Oscar et la dame rose      drama
1              2                     Cupid   thriller
2              3  Young, Wild and Wonderful      adult
3              4             The Secret Sin      drama
4              5            The Unrecovered      drama

                             movie_description  year
0  Listening conversation doctor parents , 10-yea…  2009
1  brother sister past incestuous relationship cu…  1997
2  bus empties students field trip Museum Natural…  1980
3  help unemployed father make ends meet , Edith …  1915
4  film 's title refers un-recovered bodies groun…  2007
```

[6]:
```
# Function to perform lemmatization
def perform_lemmatization(text):
    tokens = word_tokenize(text)
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(lemmatized_tokens)
```

```
# Apply stemming and lemmatization to the 'Plot' column
df['movie_description'] = df['movie_description'].apply(perform_lemmatization)

print(df.head())
```

```
   serial_number                 movie_title       genre  \
0              1      Oscar et la dame rose      drama
1              2                      Cupid   thriller
2              3   Young, Wild and Wonderful      adult
3              4             The Secret Sin      drama
4              5            The Unrecovered      drama

                              movie_description  year
0  Listening conversation doctor parent , 10-year…  2009
1  brother sister past incestuous relationship cu…  1997
2  bus empty student field trip Museum Natural Hi…  1980
3  help unemployed father make end meet , Edith t…  1915
4  film 's title refers un-recovered body ground …  2007
```

[8]:
```
# Step 2: Data Splitting
X_train, X_test, y_train, y_test = train_test_split(df['movie_description'],␣
 ↪df['genre'], test_size=0.3, random_state=42)

# Step 3: Feature Extraction
tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as␣
 ↪needed
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

[ ]:
```
# Training using Random Forest
rf_classifier = RandomForestClassifier(n_estimators = 1000, random_state = 42)
rf_classifier.fit(X_train_tfidf, y_train)

y_pred_rf = rf_classifier.predict(X_test_tfidf)
accuracy_rf = metrics.accuracy_score(y_test, y_pred_rf)
precision_rf= metrics.precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = metrics.recall_score(y_test, y_pred_rf, average='weighted')
f1_rf= metrics.f1_score(y_test, y_pred_rf, average='weighted')

print(f"Random Forest - \n Accuracy: {accuracy_rf:.2f}, Precision:␣
 ↪{precision_rf:.2f}, Recall: {recall_rf:.2f}, F1 Score: {f1_rf:.2f}")
```

```
Random Forest -
 Accuracy: 0.50, Precision: 0.50, Recall: 0.50, F1 Score: 0.41

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
```

```
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
# Training using Multinomial naive bayes
mnb_classifier = MultinomialNB()
mnb_classifier.fit(X_train_tfidf, y_train)

mnb_pred = mnb_classifier.predict(X_test_tfidf)
accuracy = metrics.accuracy_score(y_test, mnb_pred)
precision = metrics.precision_score(y_test, mnb_pred, average='weighted')
recall = metrics.recall_score(y_test, mnb_pred, average='weighted')
f1 = metrics.f1_score(y_test, mnb_pred, average='weighted')

print(f"MultiNomialNB - \n Accuracy: {accuracy:.2f}, Precision: {precision:.
 ↪2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}\n")
```

```
MultiNomialNB -
 Accuracy: 0.52, Precision: 0.49, Recall: 0.52, F1 Score: 0.43


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[10]:
```
# Training using support vector machines
svm_classifier = SVC(kernel='sigmoid')
svm_classifier.fit(X_train_tfidf, y_train)

y_pred_svm = svm_classifier.predict(X_test_tfidf)
accuracy_svm = metrics.accuracy_score(y_test, y_pred_svm)
precision_svm = metrics.precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = metrics.recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = metrics.f1_score(y_test, y_pred_svm, average='weighted')

print(f"SVM - \n Accuracy: {accuracy_svm:.2f}, Precision: {precision_svm:.2f},␣
 ↪Recall: {recall_svm:.2f}, F1 Score: {f1_svm:.2f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
SVM -
 Accuracy: 0.58, Precision: 0.55, Recall: 0.58, F1 Score: 0.55
```