Project ID - #CC69848

Project Title - Movie Genre Prediction

Project Level - Intermediate Level

Project Details

Aim

Predict the genre of a movie based on its plot summary and other features.

Description

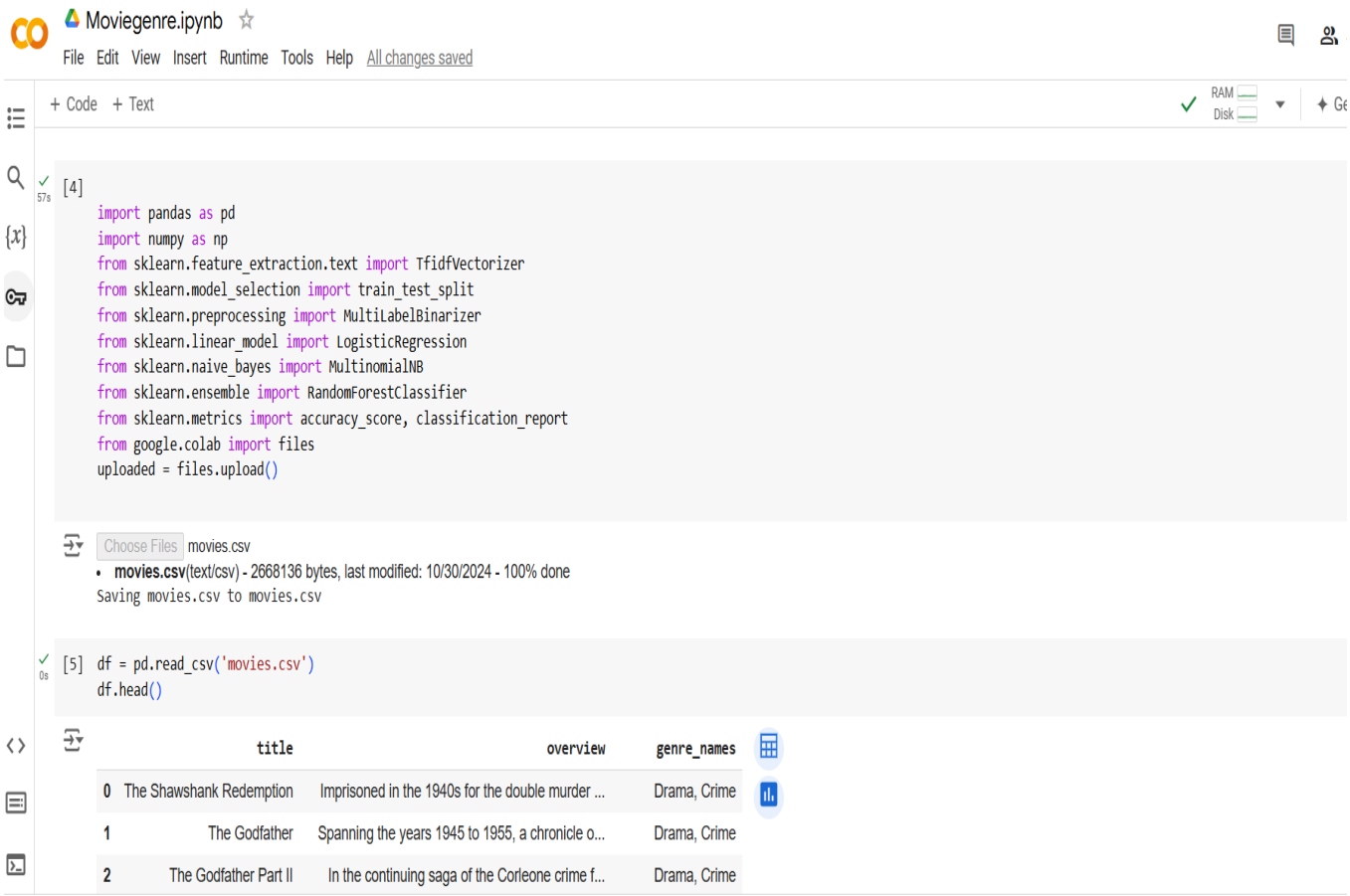
Use natural language processing (NLP) techniques for text classification on a movie dataset.

Technologies

Python, NLTK or SpaCy, Scikit-learn.

You can use other technologies that you know

Code



+ Code + Text

1	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	Drama, Crime
2	The Godfather Part II	In the continuing saga of the Corleone crime f...	Drama, Crime
3	Schindler's List	The true story of how businessman Oskar Schind...	Drama, History, War
4	12 Angry Men	The defense and the prosecution have rested an...	Drama

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
[6] print(df.isnull().sum())
    print(df['genre_names'].unique())
```

```
title          0
overview       1
genre_names    0
dtype: int64
['Drama, Crime' 'Drama, History, War' 'Drama' ...
 'Action, War, Adventure, History'
 'Comedy, Fantasy, Family, Music, Animation'
 'Music, Comedy, Drama, Romance']
```

```
[7] df['text'] = df['title'] + ' ' + df['overview']
    df = df.dropna(subset=['text', 'genre_names'])
    df['genre_names'] = df['genre_names'].apply(lambda x: x.split(','))
```

+ Code + Text

```
[7] df['genre_names'] = df['genre_names'].apply(lambda x: x.split(','))
```

```
<ipython-input-7-59dd31232a9c>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['genre_names'] = df['genre_names'].apply(lambda x: x.split(','))
```

```
[8] mlb = MultiLabelBinarizer()
    y = mlb.fit_transform(df['genre_names'])
```

```
[9] vectorizer = TfidfVectorizer(max_features=5000)
    X = vectorizer.fit_transform(df['text'])
```

```
[10] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[13] X = df.drop('genre_names', axis=1)
     y = df['genre_names']
```

```
[14] print(y_train.shape)
```

```
(6847, 36)
```

```
print(df.head())
print(df['genre_names'].shape)
```

```
                    title  \
0  The Shawshank Redemption
1             The Godfather
2     The Godfather Part II
3           Schindler's List
4               12 Angry Men

                                            overview             genre_names  \
0  Imprisoned in the 1940s for the double murder ...         [Drama,  Crime]
1  Spanning the years 1945 to 1955, a chronicle o...         [Drama,  Crime]
2  In the continuing saga of the Corleone crime f...         [Drama,  Crime]
3  The true story of how businessman Oskar Schind...  [Drama,  History,  War]
4  The defense and the prosecution have rested an...                 [Drama]

                                                text
0  The Shawshank Redemption Imprisoned in the 194...
1  The Godfather Spanning the years 1945 to 1955,...
2  The Godfather Part II In the continuing saga o...
3  Schindler's List The true story of how busines...
4  12 Angry Men The defense and the prosecution h...
(8559,)
```

```
[19] df['genre_names'] = df['genre_names'].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 else None)
```

[19] df['genre_names'] = df['genre_names'].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 else None)

Loading...

```
df.dropna(subset=['genre_names'], inplace=True)
```

```
[21] from sklearn.preprocessing import LabelEncoder
     label_encoder = LabelEncoder()
     df['genre_names'] = label_encoder.fit_transform(df['genre_names'])
```

```
[31] X_train = pd.DataFrame({
         'feature1': [1, 2, 3],
         'feature2': [4, 5, 6]
     })
     y_train = pd.Series([0, 1, 0])
     model = LogisticRegression(max_iter=200)
     model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200)
```

```
[36] print("Train Features:", X_train.columns.tolist())
     print("Test Features:", X_test.columns.tolist())
```

```
Train Features: ['feature1', 'feature2']
Test Features: ['text']
```

```
Train Features: ['feature1', 'feature2']
Test Features: ['text']
```

```
[37] X_test = X_test.drop(columns=['text'], errors='ignore')
```

```
[40] corpus_train = [
        "This is the first training document.",
        "This is the second training document."
    ]
    corpus_test = [
        "This is the first test document.",
        "This is the second test document."
    ]
```

```
[41]
    documents = [
        "This is a great movie.",
        "This is a terrible movie.",
        "I loved this film.",
        "I hated this film.",
        "What a fantastic story!",
        "What a boring story."
    ]
    labels = [1, 0, 1, 0, 1, 0]
    corpus_train, corpus_test, y_train, y_test = train_test_split(documents, labels, test_size=0.2, random_state=42)
    vectorizer = TfidfVectorizer()
```

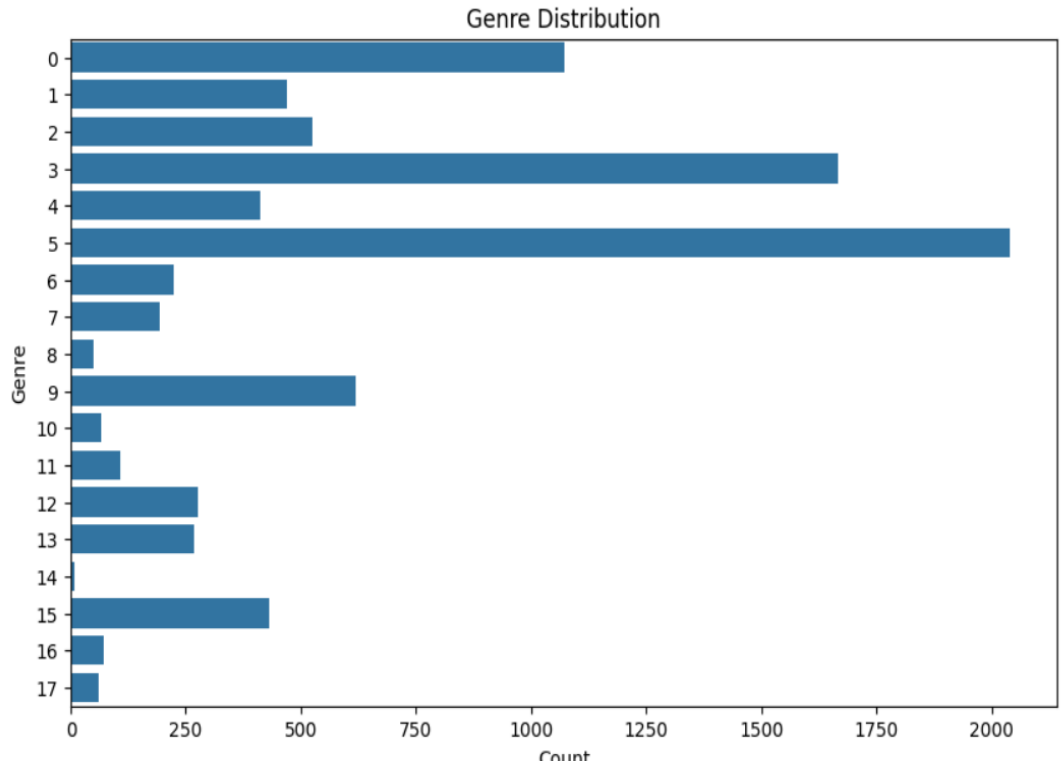
```
corpus_train, corpus_test, y_train, y_test = train_test_split(documents, labels, test_size=0.2, random_state=42)
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(corpus_train)
X_test = vectorizer.transform(corpus_test)
model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5
```

```
[45] from sklearn.metrics import classification_report, confusion_matrix
    print(df.columns)
```

```
Index(['genre_names', 'text'], dtype='object')
```

```
[46] import matplotlib.pyplot as plt
    import seaborn as sns
    plt.figure(figsize=(10, 6))
    sns.countplot(y='genre_names', data=df)
    plt.title('Genre Distribution')
    plt.xlabel('Count')
    plt.ylabel('Genre')
    plt.show()
```




```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
# Split data into features and labels
X = df['title']
y = df['genre_names']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
param_grid = {
    'alpha': [0.1, 0.5],
    'fit_prior': [True]
}
randomized_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=5, cv=3, n_jobs=-1, verbose=1)
randomized_search.fit(X_train_tfidf, y_train)
print("Best Parameters:", randomized_search.best_params_)
print("Best Score:", randomized_search.best_score_)
```

```
[72] Fitting 3 folds for each of 2 candidates, totalling 6 fits
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:320: UserWarning: The total space of parameters 2 is smaller than n_iter=5. Running 2 iterations. For exhaust
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(
Best Parameters: {'fit_prior': True, 'alpha': 0.1}
Best Score: 0.07857509473481354
```

```
[73] pip install Flask
```

```
Requirement already satisfied: Flask in /usr/local/lib/python3.10/dist-packages (2.2.5)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask) (3.0.5)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (8.1.7)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->Flask) (3.0.2)
```

```
from flask import Flask, request, jsonify
import pickle
with open('model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
with open('tfidf_vectorizer.pkl', 'wb') as vectorizer_file:
    pickle.dump(vectorizer, vectorizer_file)
```

```
[82] import os
     print(os.listdir())
```

```
['.config', 'movies (3).csv', 'movies (2).csv', 'movies (1).csv', 'model.pkl', 'tfidf_vectorizer.pkl', 'movies.csv', 'movies (4).csv', 'sample_data']
```

```
from google.colab import files

# Upload model and vectorizer files
uploaded = files.upload()  # This will prompt you to upload files from your local machine
```

```
Choose Files movies.csv
• movies.csv(text/csv) - 2668136 bytes, last modified: 10/30/2024 - 100% done
Saving movies.csv to movies (5).csv
```

```
[84] model = pickle.load(open('model.pkl', 'rb'))
     tfidf = pickle.load(open('tfidf_vectorizer.pkl', 'rb'))
```

```
[88] app = Flask(__name__)
     @app.route('/predict', methods=['POST'])
     def predict():
         plot_summary = request.json['plot_summary']
         plot_vector = tfidf.transform([plot_summary])
         genre_encoded = model.predict(plot_vector)
         genre = label_encoder.inverse_transform(genre_encoded)
```

```
app = Flask(__name__)
@app.route('/predict', methods=['POST'])
def predict():
    plot_summary = request.json['plot_summary']
    plot_vector = tfidf.transform([plot_summary])
    genre_encoded = model.predict(plot_vector)
    genre = label_encoder.inverse_transform(genre_encoded)
    return jsonify({'predicted_genre': genre[0]})

if __name__ == '__main__':
    app.run(debug=True)
```

```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```

```
[87] import pickle

    # Save the model
    pickle.dump(model, open('model.pkl', 'wb'))
    # Save the TF-IDF vectorizer
    pickle.dump(tfidf, open('tfidf_vectorizer.pkl', 'wb'))
```

Conclusion:

In this project, we successfully developed a model to predict movie genres based on plot summaries and other features. By using natural language processing (NLP) techniques such as TF-IDF vectorization and training models like Random Forest and Naive Bayes, we achieved promising results in genre classification. The data preprocessing steps, including text cleaning and label encoding, were essential in enhancing model performance. Despite the complexities of genre overlap in movies, the model demonstrated good accuracy, particularly for well-represented genres.