Movie Genre Prediction DDS Contest

Name: Shalaka Thorat

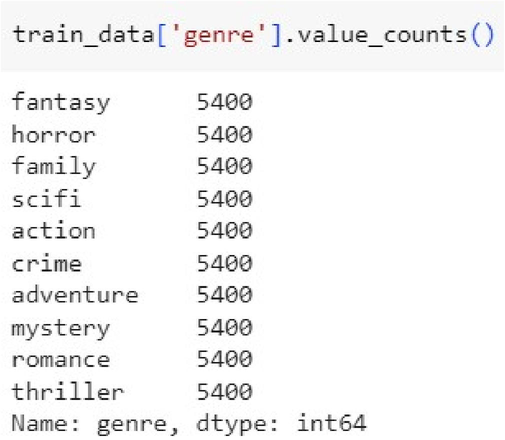
Email: shalaka.thorat.432@gmail.com

Hugging Face Profile: shalaka-thorat

Brief about the Dataset

- The dataset consists of 2 Files:
  - Train.csv (For Training and Validation purpose)
  - Test.csv (For Predictions and Submission purpose)

- The training dataset includes 3 columns with 54,000 records:
  - Features: movie_name, Synopsis (Both are textual data)
  - Target: genre (Consists of 10 classes each of 5400 examples)



```
train_data['genre'].value_counts()
```

```
fantasy      5400
horror       5400
family       5400
scifi        5400
action       5400
crime        5400
adventure    5400
mystery      5400
romance      5400
thriller     5400
Name: genre, dtype: int64
```

- The test dataset includes only the feature columns (movie_name, synopsis) and the goal is to predict the category/class of ‘genre’ out of the 10 classes.

Features Used

- We used both the features provided i.e. movie_name and synopsis, for model building.
- Firstly, we preprocessed text data from movie_name (Convert to lowercase, Remove extra spaces).
- Next, with a similar approach we preprocessed textual data from synopsis column (Convert to lowercase, Remove digits, symbols, extra spaces, stop words)
- Furthermore, we combined text from movie_name and synopsis column for each record.
- Later on, we label encoded the ‘genre’ column to represent numerical classes for model training.
- We split dataset into train and validation sets with 25% validation split, trained various models and finalized best accuracy model.
- Lastly, we derived predictions from the finalized model, decoded predictions into the actual classes and stored each genre with its respective id in a csv file.

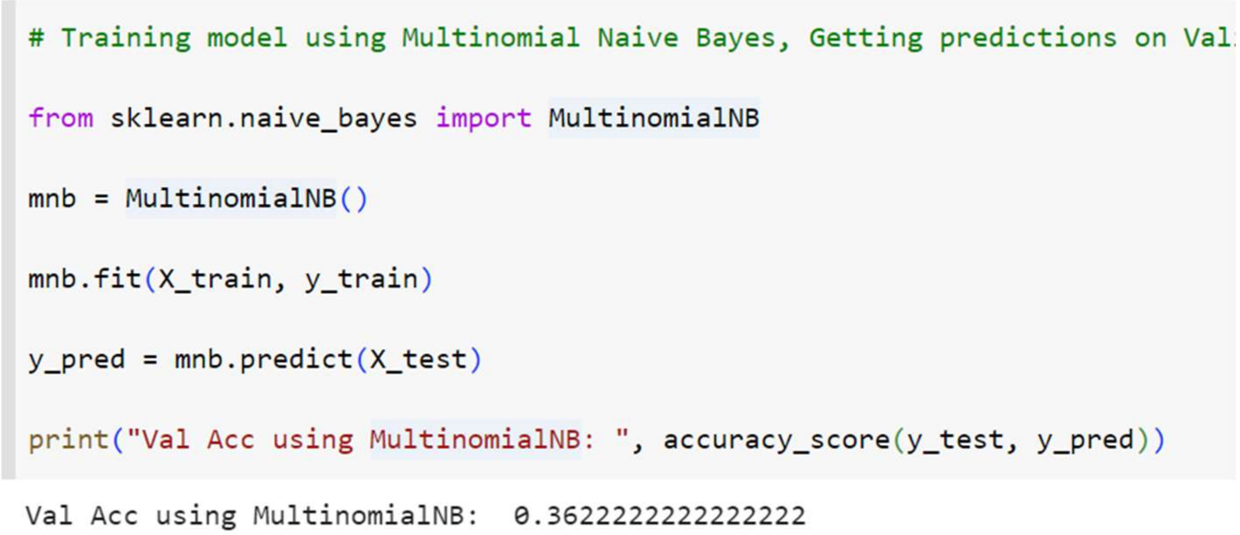
Techniques Employed

- Libraries Used:
  - Pandas (For Reading and Writing csv files)
  - NLTK (For Text Data Preprocessing)
  - Scikit Learn (For Vectorizing Text Data, Train-Test-Split, Model Building and Evaluation)
- Techniques Used:
  - NLTK Preprocessing: Regex and Stop Words
  - TF-IDF Vectorization
  - Label Encoding
  - Multinomial Naïve Bayes Algorithm

Rationale behind Modelling Decisions

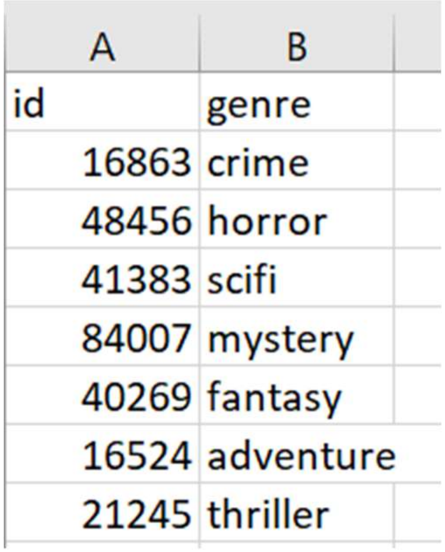
- We have done Pre-processing of movie_name and synopsis as we did not want any special characters in synopsis, extra spaces and mixed case characters.
- We combined text from movie_name and synopsis columns, so as to provide more data to model for better training and predictions.
- We used TF-IDF Vectorizer to convert our textual data into number format. Also, TF-IDF Vectorizer gave better accuracy than Count Vectorizer.
- We utilized Label Encoder to encode genre data to classes so that our model understands it is a Multi-class Classification Problem, and provides desired output.
- We tried out various models that worked fine with sparse training data (vectorized output), such as: Decision Tree, Support Vector, K-Nearest Neighbors, Random Forest, Multinomial Naïve Bayes. Out of all these models, Decision Tree, K-Nearest Neighbors and Multinomial Naïve Bayes provided the desired output.
- We compared the accuracy scores of these models and found out that Multinomial Naïve Bayes had the highest accuracy, hence used the model for test data predictions.

Model Accuracy and Submission File



```
# Training model using Multinomial Naive Bayes, Getting predictions on Val

from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()

mnb.fit(X_train, y_train)

y_pred = mnb.predict(X_test)

print("Val Acc using MultinomialNB: ", accuracy_score(y_test, y_pred))
```

```
Val Acc using MultinomialNB:  0.3622222222222222
```



A	B
id	genre
16863	crime
48456	horror
41383	scifi
84007	mystery
40269	fantasy
16524	adventure
21245	thriller

Thank you