TMDB top 5000 Film Primary Genre Prediction using Classification Models

Hamza Khattak
Dept. of Computer Science
Eastern Kenutcky University
Richmond, Kentucky
muhammad_khattak@mymail.eku.edu

Enock Kipchumba
Dept. of Computer Science
Eastern Kenutcky University
Richmond, Kentucky
enock_kipchumba@mymail.eku.edu

Isaiah Thompson
Dept. of Computer Science
Eastern Kenutcky University
Richmond, Kentucky
isaiah_thompson6@mymail.eku.edu

Bett Kiprotich
Dept. of Computer Science
Eastern Kenutcky University
Richmond, Kentucky
kiprotich_bett@mymail.eku.edu

Abstract – There has been a lot of development and increase of data in the film industry with the release of a vast array of movies produced and distributed to the audience worldwide. This project focuses on the prediction of movie genres using machine learning techniques. The objective is to develop different models that accurately predict the genre of a movie based on its various attributes, including metadata and textual information, and come up with the best model that has a high accuracy. This study utilizes the TMDB 5000 movie dataset from The Movie Database that has about 23 features including, title, budget, genre, keywords, popularity, homepage, and other relevant features [1]. By accurately predicting movie genres, the best model can assist users in discovering relevant content search and improve the overall experience of watching a movie.

I. Introduction

In recent years, the film industry has witnessed a continued increase in the volume of available content and there is a need for an effective system that can perform recommendation based on the nature of the content available. Genre prediction serves as one way of categorizing contents that are closely related and of interest to a particular group of audience. Our aim is to explore different models to accurately predict genres based on different attributes. To achieve this, we used different models including KNN, Logistic Regression, Neural Network, Random Forest Classifier, XGC Boost Classifier and Support Vector Machine Classifiers. By leveraging the rich metadata available for this movie, the models can predict and place each on different genres, with different accuracies depending on the model used. The effectiveness of the models is gauged by the accuracy score metric.

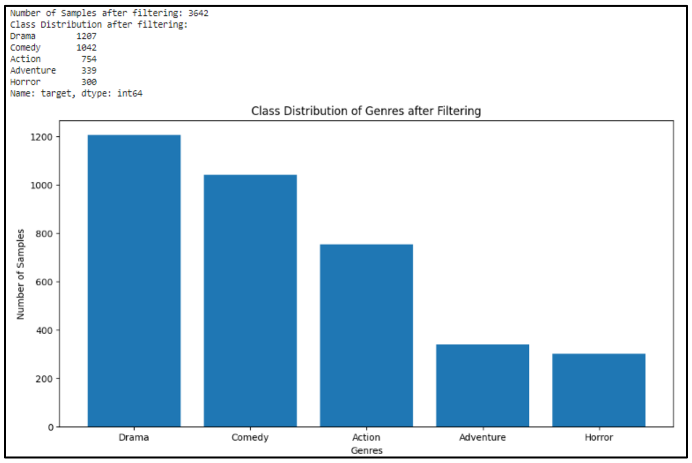
II. Data Preprocessing

Data preprocessing involved data cleaning and the transformation of raw data to be suitable for use by the models. The aim for data preprocessing is to help improve the quality of the data thus ensuring the accuracy and reliability of the predictive models.

We encountered some problems during data preprocessing. First, the dataset had too many classes leading to an imbalanced class distribution, where some classes have significantly more instances than others, this affects accuracy of the model and led to challenges later in the project. Second, the dataset had too many categorical features which increased the dimensionality and complexity of features. This also means that some of the models that are unable to deal with categorical variables will not be trained correctly. Lastly, the dataset had missing values for important features, which may lead to biases due to missing values impacting the accuracy of predictions.

To achieve our goal, we undertook different data preprocessing steps to prepare the dataset for predicting the genre of movies using multiple models. We imputed the missing values with ‘unknown’ or ‘0’ based on the data type. To ensure class balance, we excluded classes with fewer than

200 target values from the dataset. This involved filtering out the genres associated with such classes while maintaining instances from classes with enough representation. The different genres like drama had 1207 instances, comedy had 1042 instances and action had 754 instances, the rest had less than 350 instances.



```
Number of Samples after filtering: 3642
Class Distribution after filtering:
Drama        1207
Comedy       1042
Action        754
Adventure     339
Horror        300
Name: target, dtype: int64
```

Figure 1: Preprocessing for class imbalance issue

The categorical features were encoded using scikit-learn built-in function, Label encoder [2]. This involved assigning unique integers to each category in the features. Each category is mapped to a numerical value, maintaining the ordinal relationship between features. We analyzed the relationship between the numerical features in the dataset using built-in function from the seaborn library [3]. The

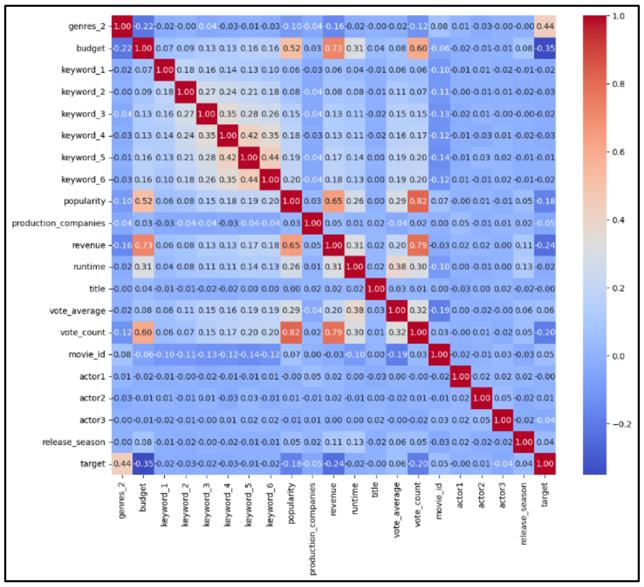


Figure 2: Correlation Matrix to avoid Multi-collinearity

function provides a correlation matrix that gives insights into how features are related to each other.

The visualization showed that features are less related to each other and by extension of the correlation matrix, multi-collinearity will not be concern proceeding forward.

In order to scale the data since different features had different scales, the dataset was normalized using the scikit-learn built-in function, Min-Max Scaler. Alternatively, Standard Scaler was also used, but the model performed visibly better using Min-Max scaling, to ensure uniformity of features.

We formatted the release dates feature into quarters, through feature engineering to use release quarters as part of the input features. Moreover, we tried out applying Principal Component Analysis (PCA) to reduce the dimensionality of numerical features, however, the PCA did not lead to significant changes in results and was not taken into consideration for the final model.

The dataset was split into three sets: training set, development set, and testing set. The dataset was split into 70% training, 15% development and 15% testing. In addition, the features and labels were extracted from the sets to facilitate predictions for training and evaluations.

The data preprocessing steps outlined resulted in a clean and transformed dataset for predictive models. By removing the missing values, label encoding features, applying class restriction, feature scaling and splitting the dataset, the steps helped improve and strengthen the foundation of our predictive models.

III. KNN IMPLEMENTATION

As previously mentioned, we put the dataset through a series of different classification models to determine the most accurate one for the dataset, all of which were developed and tested in python on Google Colab. Out of the six mentioned, KNN, Logistic Regression, and Feedforward Neural Network were the primary models we considered.

KNN, or K-Nearest Neighbor, determines the class of an input based on the classes of the nearest neighbors, with the majority deciding the label of the input [4]. The number of neighbors considered for an input is determined by the variable ***k***, which is determined by the user when the model is initialized. The optimal ***k*** value for KNN is an intricate step because a ***k*** value too small can result in overfitting and vice versa. To determine the optimal value of ***k***, we used the development set to test the model for ***k*** **-** values between 1 to 101 in steps of 4. As the figure below suggests, we found the optimal ***k*** **-**value to be between 93 and 97, and so we decided to let ***k*** = 97 for the test set. In terms of the distance used, the project utilized all three common distance metrics i.e. Euclidean, Chebyshev and Manhattan. In terms of

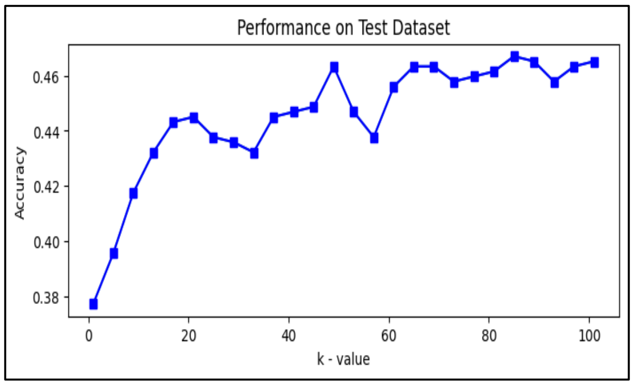


Figure 3: Finding the optimal k value using validation data

performance, the Manhattan distance turned out to be the most efficient and accurate. The accuracy score of the model on the test set was 40.58%

IV. Logistic Regression

Logistic Regression models the log-odds of an event as a linear combination of a series of independent variables derived from the input, and we are using said linear combinations to determine the classification of the input [5]. While not explicitly set, it is assumed that the decision boundary of the logistic regression is at the default 0.5 between each class, meaning that 1.5 is the decision boundary between class 1 and 2, and so on [6]. The model ran the input through 2000 iterations with a learning rate of 0.001, or 1e-3. The accuracy score of this Logistic Regression model, when used on the test set, came to approximately 42.3%. While we noticed some improvement over the KNN model, it is evident that the project may benefit from using a more complex model. Which is primarily why a Feedforward Neural Network was introduced.

V. Neural Networks

The Feedforward Neural Network is a type of Neural Network in which the data moves in a singular direction, i.e. forward [7]. The model architecture includes building blocks in a sequential stack, with 5 dense layers, each of which is entirely connected. Each batch layer, except for the last one is followed by a normalization layer, used to normalize the output of the previous dense layer. Followed by dropout activation function that is used to introduced regularization to the model so it can experiment with different weights of the input. For the compilation of the model, three commonly used optimizers were used, namely **Adam**, **RMSprop**, and **SGD.** Adam optimizer performed the best, thus it was left as a part of the final model. A learning rate of 1.0, 1e-2, and 1e-3 were used but 1e-3 was selected as the final learning rate. For the loss function, both **CrossEntropyLoss** and **sparse_categorical_crossentropy,** however **sparse_categorical_crossentropy** performed between and was used in the final model [8]. Here, the training sets was 78% of the entire dataset and the remaining was allocated to testing. The number of Epoch was selection to 200, which yielded a validation set accuracy of 66.15% at a loss of loss 0.8833. This was a significant improvement over the KNN and the multi-class logistic regression classification. The testing accuracy for the Neural Network was 49.79%.

While the accuracy had improved quite significantly from the start of the project at around 10% to all the way at 49.79%, the team realized a need for an ensemble of other classification models as well. For this purpose, the SVM, Random Forest, and XGBoost classification models were used.

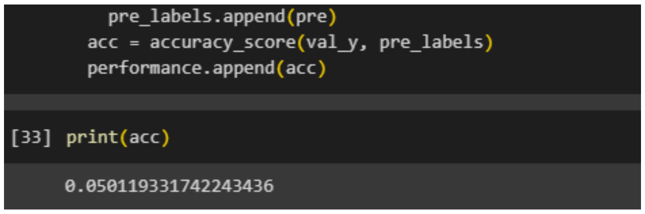
VI. SVM, Random Forest, XGBoost Models

The library Scikit-Learn was used to implement the Support Vector Machine classifier and the Random Forest classifier. The Kernel used for SVC was ‘linear’, with a regularization parameter of 0.5 and scale of gamma, leading to an accuracy score of 48.01%. For the Random Forest classifier, n_estimators of 160 was used, resulting in an accuracy score of 60.22%. The XGBoost classifier was implemented using num_class of 160, leading to an accuracy score of 61.32%. The table below represents the models using in the genre prediction project with their respective accuracy scores.

Model	Accuracy Score
KNN	40.58%
Logistic Regression	42.3%
SVC	48.01%
Neural Network	49.79%
Random Forest	60.22%
XGBoost	61.32%

VII. Result Comparison and Conclusion

Although the KNN model was able to achieve a decent Mean Absolute Error of 0.83 and Mean Squared Error of 1.51, the project's primary evaluation metric was accuracy score. The accuracy score is a performance evaluation metric from the Scikit Learn library. Initially the KNN suffered from high levels of inaccuracies as show below in figure 4.



```
        pre_labels.append(pre)
      acc = accuracy_score(val_y, pre_labels)
      performance.append(acc)

[33] print(acc)

     0.050119331742243436
```

Figure 4: Initial run of KNN for the Film Genre Prediction

However, after making the changes mentioned in data preprocessing, the group was able to find tune the model and increase the overall accuracy to 61.32%, which is a remarkable feat. While it is not easy to figure out why the model didn't achieve accuracy scores in the high 70 or 80 percents, the team believes this might be to the inert nature of the dataset and the mix of features selected to developed the models. However, the team-members are proud of their contribution and work on this successful project.

VIII. References

[1] (TMDb), The Movie Database. "TMDB 5000 Movie Dataset." Kaggle, September 28, 2017. https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata.

[2] "Sklearn.Preprocessing.LabelEncoder," scikit, accessed March 7, 2024, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[3] "User Guide and Tutorial#," User guide and tutorial - seaborn 0.13.2 documentation, accessed March 7, 2024, https://seaborn.pydata.org/tutorial.html.

[4] "What Is the K-Nearest Neighbors Algorithm?" IBM. Accessed March 7, 2024. https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20(KNN)%20algorithm%20is%20a%20non,of%20an%20individual%20data%20point.

[5][6] "What Is Logistic Regression?" IBM. Accessed March 7, 2024. https://www.ibm.com/topics/logistic-regression.

[7] DeepAI. "Feed Forward Neural Network." DeepAI, May 17, 2019. https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network.

[8] Tf.Keras.Losses.SparseCategoricalCrossentropy: Tensorflow V2.15.0.POST1." TensorFlow. Accessed March 7, 2024. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy.

Topic	Author
Abstract & Part I	Bett Kiprotich
Part II	Enoch Kipchumba
Part III & Part IV	Isaiah Thompson
Part V, VI, VII	Hamza Khattak