Model Design

Farida Bey
900212071

&

Youssef Ghaleb
900211976

INTRODUCTION

Given the multi-label, text-heavy data, and the mixed performance of Random Forest and Decision Trees, we concluded last time that a Neural Network model might offer the best capacity to handle complex, non-linear patterns, especially with further tuning. However, due to its low metrics in less common classes we want to focus on improving those metrics using techniques like *focal loss*, which addresses class imbalance by applying higher penalties to misclassified infrequent classes.

MODEL DESIGN

1. Architecture Design:

We will design a Neural Network with the following architecture:

1. ***Input Layer:*** The input layer will consist of 1515 features, which include both numerical features and embeddings for text features like movie titles, descriptions, and keywords.
2. ***Hidden Layers:***
   i. *The first hidden layer* will have 512 units with a ReLU activation function, allowing the network to capture complex relationships.
   ii. *The second hidden layer* will have 256 units, continuing the refinement of learned representations.
   iii. *The third hidden layer* will have 128 units, learning deeper abstract features.
3. ***Dropout Layers:*** Dropout will be applied after each hidden layer to prevent overfitting.
4. **Output Layer:** The output layer will have 19 units (corresponding to the 19 genres) with a *sigmoid activation* to predict probabilities for each genre in a multi-label classification setup.

This architecture is chosen for its ability to handle large and complex feature spaces while also ensuring the model doesn't overfit the training data.

2. Regularization Techniques:

To prevent *overfitting*, we will apply the following regularization techniques:

Dropout*:* Dropout layers will be applied after each hidden layer to ensure that the network doesn't rely too heavily on any single neuron.

L2 Regularization: We will be researching adding L2 regularization to the dense layers to penalize large weights and reduce model complexity, so that the models generalize.

3. Loss Function:

We will use *focal loss* instead of standard binary cross-entropy to address the class imbalance problem. Focal loss focuses on harder-to-classify examples, particularly those from infrequent genres, by applying a higher penalty to misclassified examples.

4. Class Weights:

To handle class imbalance, we will compute **class weights** based on the inverse frequency of genres. This will ensure that underrepresented genres are given more importance during training. The class weights will be used to adjust the loss function during model training.

5. Activation function:

ReLU Activation*:* The hidden layers will use Rectified Linear Unit activation, which helps mitigate the vanishing gradient problem and speeds up convergence.

Sigmoid Activation*:* The output layer will use the sigmoid activation function, which is suitable for *multi-label classification*, allowing each genre to be predicted independently as a binary value (0 or 1).

6. Tuning and Optimization:

To refine the model's performance, we will apply the following optimization techniques:

6.1. Manual search:

We will begin by adjusting key hyperparameters such as learning rate, dropout rate, and batch size. These parameters will be fine-tuned based on the model's performance on the validation set, with a particular focus on improving metrics for underrepresented genres.

6.2. Grid search:

An exhaustive search over a predefined set of hyperparameters will be conducted to find the optimal combination. Cross-validation will be used to evaluate each combination, ensuring the model generalizes well to unseen data.

6.3. Random search:

We will implement random search to efficiently explore a broader range of hyperparameters. This will allow us to sample combinations without the computational cost of grid search.

6.4. Other Optimization Techniques

6.4.1. Early Stopping:
Training will be halted when the model's performance on the validation set no longer improves, preventing overfitting.

6.4.2. Learning Rate Schedulers:
The learning rate will be adjusted dynamically during training to improve convergence.

6.4.3. Batch Normalization:
Batch normalization will be applied to stabilize the training process and potentially improve performance.

These optimization methods, along with regularization techniques and class imbalance solutions, will help improve the model's performance, especially for less common genres.

UTILITY APPLICATION DESIGN

1. Application Architecture:

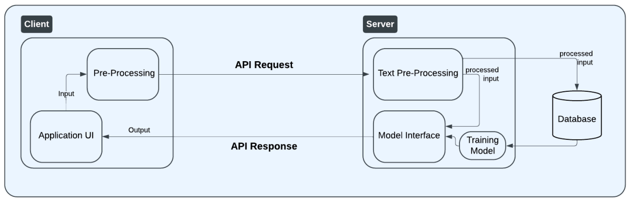

Figure 1.1 Application Client-Server Architecture

The architecture of our application is built around a ***client-server model***. This client-server architecture ensures that the system remains scalable, modular, and maintainable while offering flexibility for future upgrades (e.g., adding new features, retraining the model, or improving the UI). By isolating complex tasks to the server and focusing the client on interaction and presentation, this approach keeps the system efficient and user-friendly.

a. Client-Side Responsibilities:
The client side focuses on providing a
i. user-friendly interface (UI). It accepts user inputs such as movie titles, descriptions, and runtimes, and then displays the predicted genres for the movie. Additionally, a ***ii. preprocessing component*** is hosted on the client side, performing minimal data preprocessing. This includes basic data cleaning, such as tokenization or handling special characters, to prepare the data before it is sent to the server. Once the input data is processed, it is transmitted to the server via an ***API request***, initiating the client-server communication.

b. Server-Side Responsibilities:

Upon receiving the input data from the client, the server performs more advanced tasks to ensure accurate predictions.

i. Text Preprocessing*:* The server handles *advanced natural language processing (NLP)* techniques, which are crucial for preparing the input data in a format suitable for the model. Steps involved include:

- *Stopword removal*: Eliminating common, non-informative words that do not contribute to genre prediction.
- Stemming: Reducing words to their base or root form.
- *Word Embedding Generation*: Using a *pretrained Word2Vec model* to convert the text data into numerical vectors that capture semantic meaning. These embeddings allow the model to better understand the context and relationships within the movie descriptions.

ii. Model Interface*:* After preprocessing, the data is passed to the model interface, which is responsible for invoking the neural network model. The model uses the preprocessed data to *generate predictions*. These predictions represent *all the movie genres* that most likely correspond to the given input.

iii. Training Model: We also included a retraining component to ensure that the model stays up to date and maintains high prediction accuracy, the system supports *periodic retraining*. This involves:

- Collecting user feedback and new movie data.
- Retraining the model with updated datasets to improve its performance, especially on ***underrepresented genres.***

The retraining process can be automated to run at regular intervals (e.g., weekly or after a certain number of new data points), ensuring the model adapts to emerging trends and improves over time.

iv. Database: The database component is used to store

- **Existing Dataset**: Our existing dataset in addition to:
- **New Movie data**: User-submitted movies, their descriptions, and associated metadata (e.g., runtime).
- **Predictions**: The model's predicted genres for each movie, along with the associated confidence scores or probabilities.
- **User Feedback**: Responses to the predictions, such as whether the genres predicted were correct or not.

The database stores this information to be used by the *Training Model* for *continuous learning*, allowing the model to improve based on real user feedback and new movie data.

Once the model generates predictions, the server sends back a structured ***API response*** containing the ***predicted genres*** back to the client. The client then displays the result to the user in a clear and easily understandable format, listing the most probable genres for the movie.

2. Data Flow and Interaction:

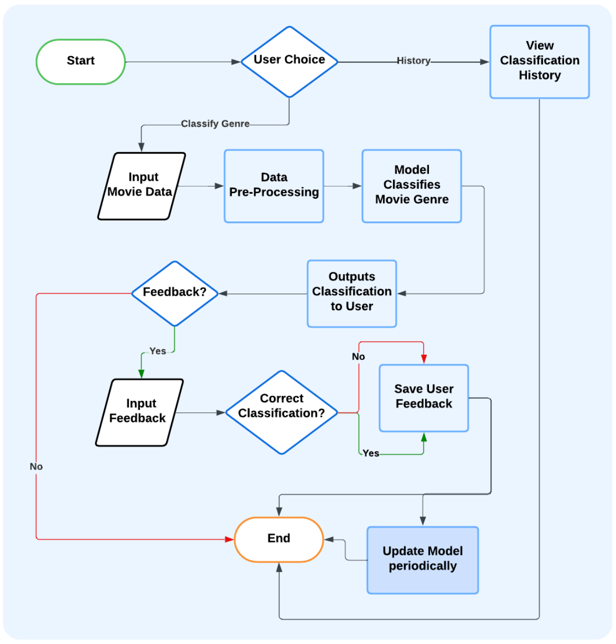

Figure 1.2 Flow Chart

The flow chart illustrates the user experience as they interact with our application. Upon entering the application, users are presented with two options: *"Classify Genre" or "History."* The diagram outlines the processes the user follows based on their selection. If the user opts to view the classification history, they are redirected to a page where all previous classifications are displayed. After viewing the history, the user's journey concludes as they have fulfilled their purpose. On the other hand, if the user selects "Classify Genre," the user is prompted to input the title of the movie and its meta data. The data is then pre-processed before being sent to the model for classification. Once the model classifies the movie into a specific genre, the result is returned to the user. At this stage, the user is given the option to provide feedback on the classification. If the user chooses to provide feedback, they are asked to specify whether the classification was accurate. Regardless of their choice, the feedback is stored and used during the application's scheduled updates, when the model will be retrained and improved based on user input. Once the feedback is provided (or if the user chooses not to provide any), their interaction with the application concludes.

3. User Interaction:

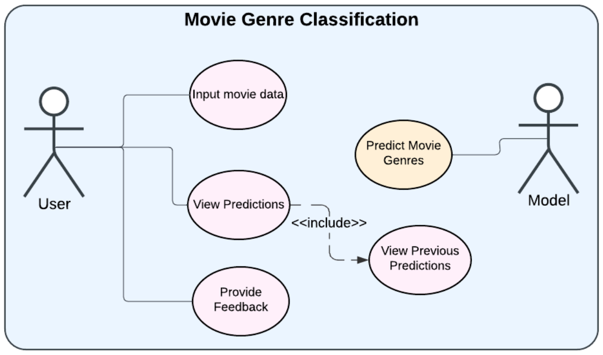

Figure 1.3 Application Use Case Diagram

The use case diagram above depicts the interactions between the user and the model powering the application.

User Actions:

- The user inputs the movie metadata.
- The user views the predicted genres after the model processes the input.
- The user can access and review past predictions.
- The user can provide feedback on the predictions.

Model Actions:

- The model processes the movie metadata input to predict the genres.
- The model is updated based on user feedback to improve future performance.