Group Project report

# Group member:

Group Name: LLP

Haocheng Lu A0276434Y E1132167

Paul David Christopher Golling A0294203L E1345974

LUO Ming A0244396W E0860682

# Data processing

A common data processing is shared between all three approaches.

The data given is in a set of CSV files. The *u.data* captures the relationship between users and movies. The *u.item* contains information about a movie. It is also associated with *u.genre,* which contains genre information and uses item ID as a foreign key. The *u.user* contains information about users. The *u.train* and *u.test* are split data from *u.data*, which will be later used in training and evaluation processes for different approaches.

First, the data file is read from CSV into a Pandas DataFrame. Each file is read into a different DataFrame. Then, the DataFrame containing *u.data* will be used as the base to merge other data frames in.

The line number is checked using *info()*methodto ensure no lines are missing after the merge.

*u.user* data and *u.item* data are merged to the *u.data* based on ‘user\_id’ and ‘item\_id’ (Pandas, n.d.). Based on the key ‘user\_id’ and ‘item\_id’, the method for merging is an inner merge, which means it will only keep the intersection part between data from *u.user* and *u.item.* This is because of the nature of the table where ‘user\_id’ and ‘item\_id’ are foreign keys in the *u.data*. After merging, the DataFrame info is checked. It can be confirmed that all columns in *u.user* and *u.item* have been merged into the DataFrame with no loss of rows.

Due to the final three approaches, there will be no temporal analysis applied as the timestamp is assumed to only bring noise. Therefore, this column is dropped. The columns “movie title,” “IMDb\_ URL,” and “video release date” are also dropped from the DataFrame. The “movie title” serves a purpose similar to ‘item\_id’, which is only an identifier for the movie and does not provide much information for further analysis. The ‘IMDb\_URL’ provides no further information that needs to be retrieved, so it is also considered non-contributory to the analysis. The ‘video release date’ column is found to be empty, and the actual release information is contained in another column called ‘release date’; hence, this column is also dropped.

To unify the release date and make the data input more generalized, only the release year is kept, and the specific month and date are dropped. Keeping the date too specific provides very little benefit but introduces unnecessary noise to the model.

Also, the genre was one-hot encoded in the original data. At this point, it is not clear how each approach will use this feature, so it is reduced to one column.

Finally, the information in ‘zipcode’ is not useful as it is only a code. We used an API to convert the zipcode into a state name. It is found that some of the zipcodes may not be U.S. zipcodes, in which case they will be marked as “Outside of USA.” Due to the process of using the API (zippopotam.us, n.d.) to convert all zip code data being time-consuming, a multi-threaded process is used to accelerate the whole process. Finally, due to the requirement to train our models using *u.train* and perform evaluation on them with *u.test*, they are also read into a Pandas DataFrame. A left merge is performed with the *u.data* DataFrame to get user information and movie information into the data for the training and test set.

# Approach 1 Neural Networks/MLP

In this approach, a Multi-Layer Perceptron is developed as a binary classifier to predict if a certain user will like or dislike a movie. The feature “like or dislike” is added to the original data based on the user’s rating of the movie. Other features, such as user information and movie information, will be used as input for the network.

First, the training and testing data are read into a Pandas DataFrame, and the dimensions and NaN values are checked. It is found that in the “release date,” there is a very small amount of data with NaN values in both the test and train data. These rows with NaN values are dropped. Then, the “user\_id” and “item\_id” columns are dropped. These two columns serve as foreign keys in the original data and do not provide any useful information. The rest of the columns are considered useful features and will be used for building the MLP.

Then the distribution of all features is checked to see if the data is imbalanced. It is found that although some features, such as “release date,” have an imbalanced distribution, the test data has a similar distribution. Hence, the effect of imbalanced data is considered negligible.

Based on the check on the distribution of ratings, the “like” and “dislike” categories are decided as follows: a rating of 4 or above will be marked as “like,” while a rating of 3 or below will be marked as “dislike.”

Starting to build the neural network. First, the train dataset and test dataset classes are defined. These classes are mainly used to perform feature engineering on the data. For example, in the train dataset class, methods for performing one-hot encoding and converting the DataFrame to a PyTorch tensor are defined. The first MLP is defined as a simple network with three layers: one input layer, one hidden layer with 128 neurons, and an output layer. The input and hidden layers use ReLU as the activation function, and the output uses a sigmoid function; a threshold will be defined later to separate the binary classes.

Then, the training function is defined, and the model and training data are initialized. The loss function is Binary Cross-Entropy, and the optimizer chosen is Adam. The model is trained with 500 epochs, but the loss decreases very slowly. After training, the model is evaluated with the test dataset. The threshold to separate the probability into “like” or “dislike” classes is calculated using the Youden index.

From the results, it can be seen that the model performs poorly on the “like” class with an F1 score of only 0.33. The classification on “dislike” is also not very strong, with an F1 score of 0.56. An ROC plot shows that the area under the curve is only 0.51, indicating that the model is almost performing like a random classifier.

图表, 折线图

描述已自动生成

Figure 0‑1 ROC plot for simple model with one-hot encoding

It is found that one-hot encoding can cause an issue in that the test dataset does not contain all the movie genres. The test data lacks “genre\_Fantasy” and “genre\_Unknown,” which causes the input tensor length to mismatch with the model input size. To solve this problem and improve model performance, word embedding is implemented.

For the “user\_zip” and “user\_occupation” columns, due to the large number of categories, it is considered appropriate to use word embedding. For the “user\_gender” column, it may be better to keep using one-hot encoding since it will only be “male” or “female.” Before “user\_zip” and “user\_occupation” are passed into the embedding layer, they are encoded with an ordinal encoder, as the embedding layer requires integers as input. Other than adding the embedding layer, the other settings of the first version of the model remain unchanged. The resulting ROC curve is shown below in Figure 0‑2. It can be observed that with the embedding layer, the overall performance of the model is slightly improved. From the classification report, the “like” class F1 score has been significantly improved from 0.33 to 0.56.

图表, 折线图

描述已自动生成

Figure 0‑2 ROC curve with embedding layer model

Based on the second version of the model, k-fold validation is used to tune the hyperparameters. The tuned hyperparameters include the embedding dimension, learning rate, number of hidden layers, number of neurons per hidden layer, and different activation functions.

# Hyperparameter grid

embedding\_dims = [8, 16]

learning\_rates = [0.005, 0.01]

num\_layers\_list = [2, 3]

num\_neurons\_list = [64, 128, 256]

activation\_functions = [nn.ReLU, nn.Tanh, nn.LeakyReLU]

K-fold validation is performed on the above hyper-parameter grid. The result is as following:

Best Hyperparameters: {'embedding\_dim': 16, 'learning\_rate': 0.005, 'num\_layers': 2, 'num\_neurons': 256, 'activation\_fn': <class 'torch.nn.modules.activation.ReLU'>}

Best F1 Score: 0.6729

With the result hyper-parameter, the model is trained and evaluated, the ROC curve is shown in the following Figure 0‑3.

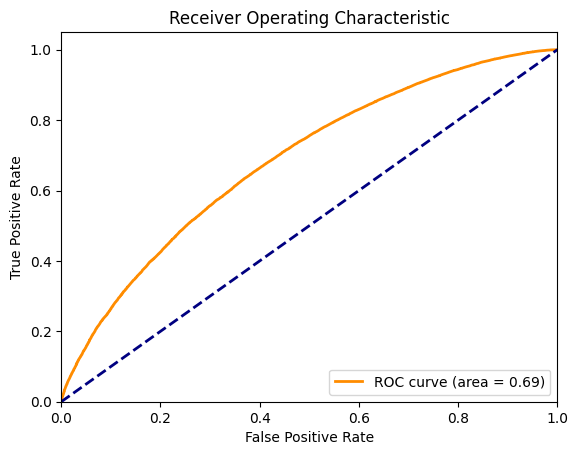


Figure 0‑3 ROC for best hyperparameter model

The model with the best hyper-parameters shows have better overall performance with ROC curve has the AUC 0.7. Additionally, the f1 score for class like and dislike has increased to 0.63 and 0.66.

In the above model, due to the time-consuming nature of k-fold validation, the training epochs are fixed at 50. To explore how the model behaves with a larger number of training epochs, 1000 epochs are used in a new run. To prevent overfitting, the model is also adjusted so that each hidden layer has half the number of neurons as the previous hidden layer. The results, shown in Figure 0‑4, indicate further improvement, with an ROC AUC of 0.8 and F1 scores for the “dislike” and “like” classes at 0.70 and 0.72, respectively.图表, 折线图

描述已自动生成

Figure 0‑4 Model training with 1000 epochs

# References:

Pandas. (n.d.). *pandas*. Retrieved 10 2024, from pandas: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html

zippopotam.us. (n.d.). *zippopotam*. Retrieved 10 2024, from zippopotam: http://api.zippopotam.us