Project 3

Rating Prediction

Choy Hill 20765307
Law Chun Yin 20771966

A COMP4332 Project Report

May 13, 2024

1 Methodology

This report aims to provide a detailed explanation of our approaches to building a rate prediction model using the neural collaborative filtering (NCF) framework. In this report, we will first explain the changes we made to the baseline NCF model provided in Tutorial 8 in order to create the best-performing model that surpasses the toughest baseline performance on the validation set. And then further elaborate on what additional approaches we have attempted further to improve the performance of our collaborative filtering model.

Our group focuses on fine-tuning the model's embedding size, the number of training epochs, the optimizer used, and the activation function used in order to evaluate whether these adjustments will improve the performance of our model.

Firstly, in regards to the embedding size and the number of training epochs used, we evaluated by training multiple models with different settings to determine which configuration would yield the best-performing model on the validation sets. The code used for this evaluation is as follows::

```
# Define the values for epochs and embed_size
epochs = [5, 10, 15, 20, 25, 30]
embed_size = [100, 150, 200, 250, 300]

# Create an empty array to store the validation RMSE values
result = []
output_data = []

# Loop through the combinations of epochs and embed_size
for num_epochs in epochs:
    row = []
    for size in embed_size:
        # Build and compile the model with the desired parameters
        model = build_ncf_model(n_users, n_items, embed_size=size, output_layer='mlp')
        model.compile(optimizer='SGD', loss='mse')

        # Fit the model on the training data
        history = model.fit(
            [re_reviewer, re_product],
            re_ratings,
            epochs=num_epochs,
            verbose=1,
            callbacks=[ModelCheckpoint('models/model.keras')])

        # Load the model and make predictions on the validation set
        model = tf.keras.models.load_model('models/model.keras')
        val_pred = model.predict([val_reviewer, val_product])

        # Calculate RMSE
        val_rmse = rmse(val_pred, val_ratings)
        row.append(val_rmse)
        output_data.append((num_epochs, size, val_rmse))
        print("Epochs: {}, Embed Size: {}, Val RMSE: {}".format(num_epochs, size, val_rmse))

    result.append(row)
```

Figure 1: The evaluation code

As for the optimizer used in the training process, we replaced the baseline Adaptive Moment Estimation (Adam) optimizer with Stochastic Gradient Descent (SGD) to assess if this change would improve the model's performance.

Before delving into our approach and the various activation functions we employed, it is important to outline the structure of the Multilayer Perceptron (MLP) layer in the NCF model. The structure is as shown below:

The MLP layer in our NCF model consists of several interconnected dense layers that perform higher-level feature extraction and capture complex patterns and interactions in

```
elif output_layer == 'mlp': # Try out different activatio function:
                            # Softmax: Avoid using softmax activation for link pre
                            # ELU: consider negative inputs, a variation of ReLU
                            # sigmoid
    # Concatenate the users' and items' embeddings as the input of MLP
    mlp_input = Concatenate()([user_emb, item_emb])
    # Perform higher-level feature extraction with 128 units
    mlp_layer_1 = Dense(units=128, activation='Relu')(mlp_input)
    # The layer with 128 units learns more abstract representations
    # by capturing complex patterns and interactions in the data
    # Perform feature transformation with 64 units
    mlp_layer_2 = Dense(units=64, activation='Relu')(mlp_layer_1)

    # The layer with 64 units further refines the learned representations
    # and captures more intricate patterns in the data

    # Perform feature transformation with 32 units
    mlp_layer_3 = Dense(units=32, activation='Relu')(mlp_layer_2)

    # The layer with 32 units continues to extract higher-level features
    # and captures more fine-grained patterns in the data

    # Generate the final prediction with 1 unit
    model_output = Dense(units=1)(mlp_layer_3)
else:
```

Figure 2: The MLP layer

the data. The input to the MLP layer is obtained by concatenating the embeddings of users and items.

The **first layer**, with 128 units, applies the Rectified Linear Unit (ReLU) activation function to transform the concatenated embeddings. This layer learns more abstract representations by capturing intricate patterns and interactions in the data.

The output of the first layer is then passed to the **second layer**, which has 64 units and also utilizes ReLU activation. This layer further refines the learned representations and captures more intricate patterns in the data.

Next, the output of the second layer is fed into the **third layer**, which has 32 units and employs ReLU activation. This layer continues to extract higher-level features and captures more fine-grained patterns in the data.

Finally, the output of the third layer is processed by a layer with 1 unit, which generates the final prediction. This prediction represents the estimated rating or preference value for a specific user-item pair based on the learned representations in the MLP layer.
For our model, we have chosen different activation functions for each layer to assess whether the model's performance can benefit from the use of diverse activation functions. In this regard, we have employed **Softmax**, **ELU**, and **ReLU** as the activation functions.

In the later stages of our testing, we also attempted to remove some extreme data points from the training dataset. This was done to evaluate whether excluding cases where items or reviewers have a disproportionately high frequency of ratings would improve the model's performance. The objective was to reduce the potential bias caused by highly frequent items or reviewers and assess if it leads to improved model performance.
In the following section, we will present the results of the above-mentioned approaches and their impact on the overall performance of the NCF model. Additionally, we will provide the results of each individual model.

2 Result

In this section, we will demonstrate the performance of each model under different settings using the root mean squared error (RMSE) as a metric to evaluate the model performance. The results will be presented in the form of a heatmap, allowing for easy comparison and analysis.

2.1 Best Result

The best-performing model was achieved with the following settings:

```
Embedding Size: 150, Epochs: 5, Verbose: 1, Validation RMSE: 0.837793
```

This model surpassed the highest baseline result in terms of rating prediction on the validation review dataset. The model was saved in the "**models/model.keras**" directory within the submitted code.

2.2 Number of Epochs and Embedding Size

In this section, we will report our findings regarding the impact of fine-tuning the embedding size and the number of training epochs on the NCF model. The following tests were conducted with a fixed setup using the SGD optimizer and the ReLU activation function.

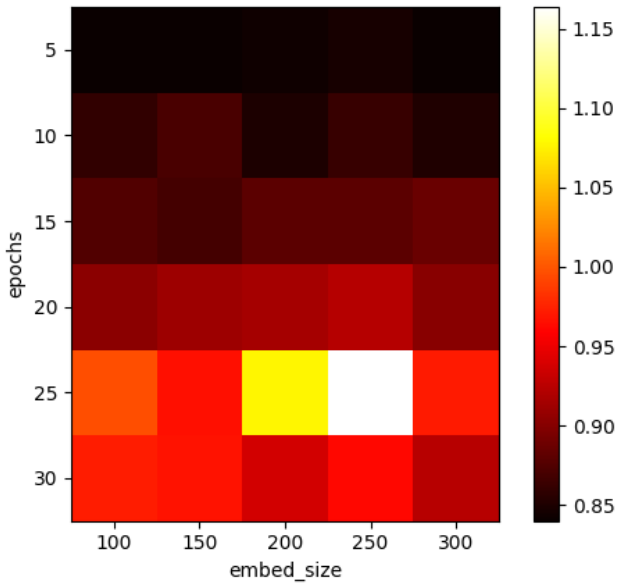


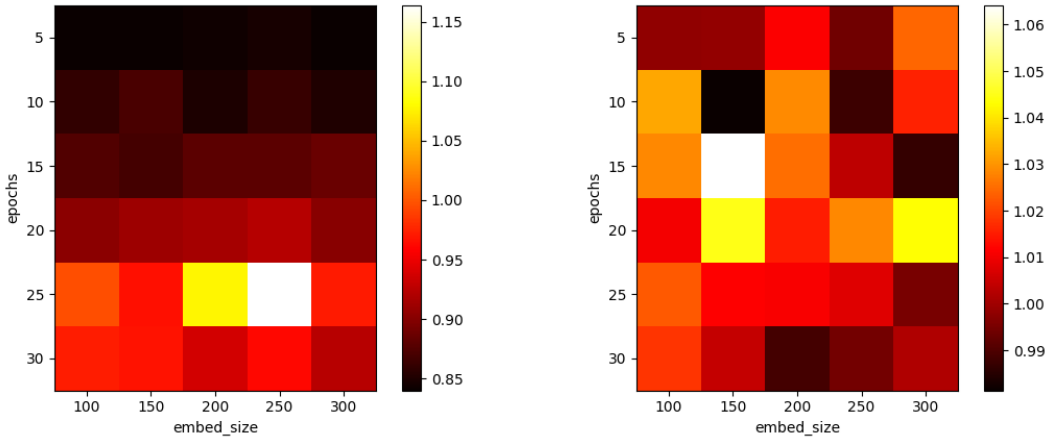
Figure 3: The Result Heatmap

As depicted in Figure 3, the performance of the resulting NCF model did not exhibit any noticeable difference with different embedding sizes. Only a marginal improvement of approximately 0.01 was observed in the validation tests. However, a smaller number of training epochs did enhance the overall performance of the model on the validation set.

We suspect that this is primarily because a larger number of training epochs can result in overfitting the model to the training dataset, thereby significantly hampering its performance on unseen data.

2.3 Use of different Optimizers

In this section, we will report our findings regarding the impact of using different optimizers during the training stage on the overall final performance of the NCF model. Similar to Section 2.2, the tests were conducted with different embedding sizes and number of epochs. However, for this section, only the activation function used was fixed to ReLU. The optimizers used and their resulting performance are as follows:



(a) With SGD

(b) With Adam

Figure 4: Result Heatmaps

As depicted in Figure 4, the overall performance of the resulting model exhibited significantly better results when trained with SGD compared to Adam. We believe this is primarily due to the following reasons:

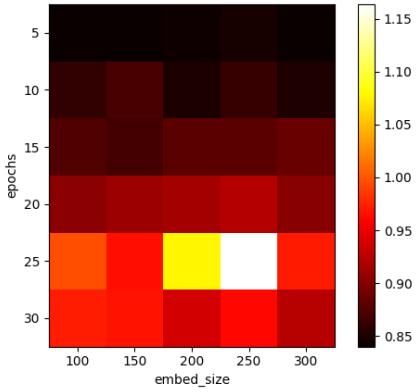
a) SGD has been observed to converge to sharp minima more effectively than Adam.

b) SGD can be more robust to noisy gradients compared to Adam. In some cases, Adam's adaptive learning rate and momentum can overly amplify the effect of noisy gradients, leading to erratic updates.

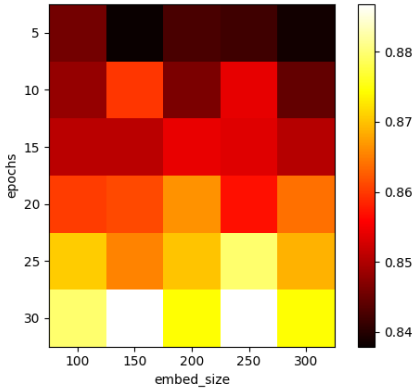
Thus, we concluded that, within the scope of this project, SGD is a better option for training an NCF model for the rating prediction task. However, during our tests, we observed that the models trained with SGD did not consistently perform well. Therefore, further testing is required to investigate the factors that may have contributed to this inconsistency. We suspect that the randomness inherent in SGD may have played a role in this variation of performance.

2.4 Use of Different Activation Functions

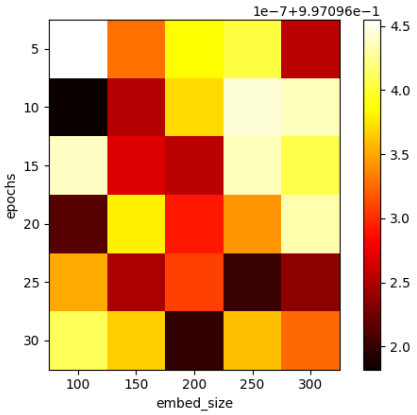
In this section, we will report our findings regarding the impact of using different activation functions for the NCF model on its overall final performance. In contrast to Section 2.3, the tests were conducted with a fixed SGD optimizer and varying embedding sizes and number of epochs. The activation functions used and their resulting performance are as follows:



(a) With ReLU



(b) With ELU



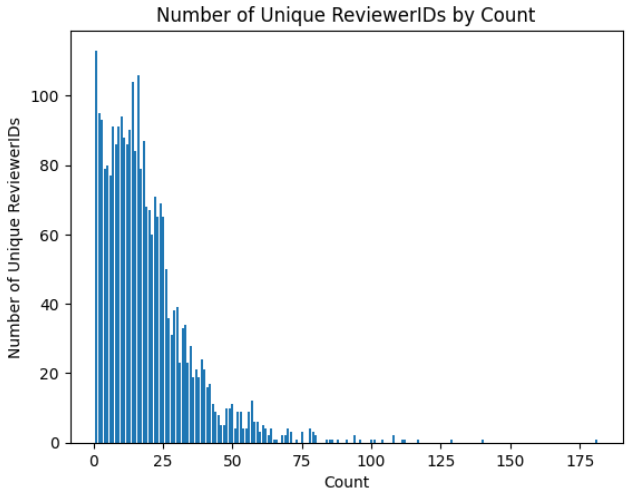
(c) With SoftMax

Figure 5: Result Heatmaps

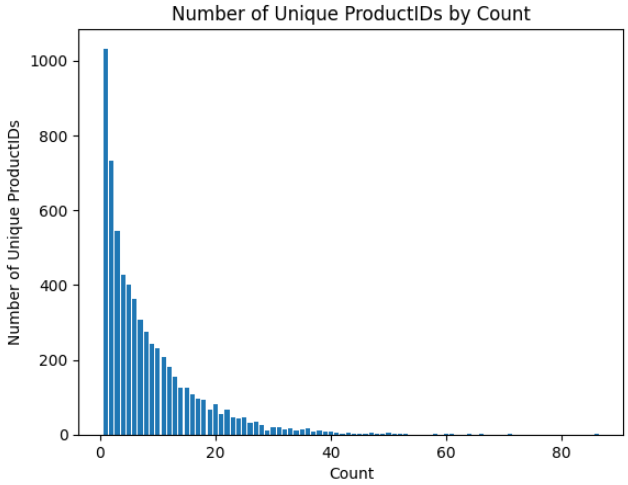
The results in Figure 5 indicate that the resulting model performed better with ReLU and ELU activation functions compared to the Softmax function. Softmax, which assigns probabilities for multi-class tasks, can lead to convergence issues and stagnant loss rates when used for link prediction. Hence, it is advisable to avoid using softmax activation for such tasks. Both ReLU and ELU are similar in terms of introducing non-linearity, addressing the vanishing gradient problem, and producing sparse activation. ReLU is computationally simpler and faster, while ELU handles negative inputs more effectively. Overall, both activation functions demonstrated comparable effectiveness in the NCF model for the given task.

2.5 Attempt to Remove Extreme Items

In this section, we will report our findings regarding the impact of removing extreme items in the training dataset. After visualizing the dataset, as shown in the following figures, we discovered the presence of extreme data points. We are interested in exploring the advantages of removing these extreme items from the dataset and how it can potentially improve the performance of the resulting NCF model.



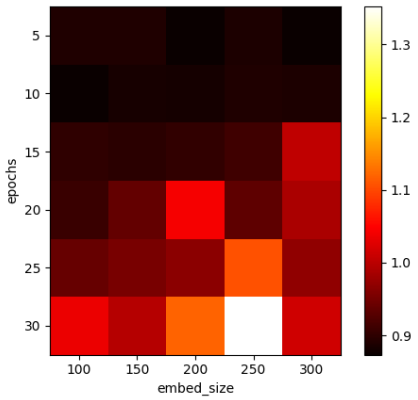
(a) Visualization of Reviewesr



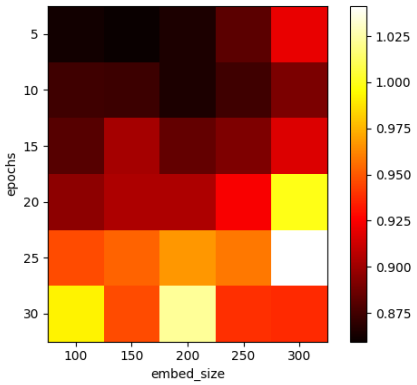
(b) Visualization of Products

Figure 6: Result Heatmaps

Similar to Section 2.2, the following tests were conducted with a fixed setup using the SGD optimizer and the ReLU activation function with different embedding size and number of epochs:



(a) Attempt 1



(b) Attempt 2

Figure 7: Result Heatmaps

In Figure 7a, we removed reviewers from the training dataset who had more than 50 reviews. However, this removal did not yield a promising improvement in the performance compared to the baseline performance of our model. However, in Figure 7b, we made another attempt by removing data with more than 50 reviews from reviewers and products that had more than 20 reviews from the dataset. The performance of the resulting model was comparable to the hardest baseline provided in the project description.. This suggests that removing some extreme data can contribute to improving the model's performance. However, further tests and studies are needed to fully understand the extent of this approach's impact on the overall performance of the model.