Personalisation and Machine Learning_ Mini Project

Korina Kyriaki Zaromytidou
ID_ 22046361

GitHub Repository for this Project: https://github.com/22046361/PML_Mini_Project_Zaromytidou/

Brief: Using an existing dataset, try and improve the quality, accuracy or fairness of recommendations or customer segmentation using a machine learning app

Objective and Approach:
For this assignment, I chose Brief 1 among the proposed options. The objective of this brief was to enhance the quality, accuracy, or fairness of rec ommendations or customer segmentation using a machine learning approach. As a beginner in machine learning and coding, my personal goal was to improve my technical skills in developing and utilizing machine learning techniques while also expanding my understanding and practical applica tion of data science concepts. I considered Briefs 1 and 2 but ultimately opted for Brief 1 as it offered a more practical project compared to Brief 4, and Brief 3 aligned closely with exercises and assignments I had previously completed. I spent considerable time searching for a suitable dataset an d tested various options before finalizing my decision. However, I acknowledge that this approach may not have been the most efficient one.

My primary objective was to develop a Collaborative Filtering model that leverages user behaviour and preferences to provide personalized book r ecommendations. By capturing user-book interactions and learning latent features through embeddings, I aimed to make predictions for unseen us er-book pairs. My goal was to generate accurate and relevant book recommendations based on user ratings. This involved data manipulation and d esigning the architecture of the model. To assess the model's performance, I used appropriate evaluation metrics.

Dataset Selection:
For my project, I selected the "Book Recommendation Dataset" obtained from Kaggle (RUCHI, 2023). This dataset contains anonymized informatio n from 278,858 users, including demographic details, along with 1,149,780 ratings given to approximately 271,379 book titles. Since the dataset foc uses on user-book interactions and ratings, I decided to use collaborative filtering as the most suitable technique for building a recommendation sy stem.

Data Cleaning and Pre-processing:
The full code for the cleaning and pre-processing can be found in the provided notebooks.

To start, I conducted a comprehensive exploration and analysis of the dataset, including data visualization and understanding the context. I carefull y examined each column to identify any missing or irrelevant information. I decided to remove ratings with missing values or a value of 0. This decis ion was based on the lack of specific information regarding whether a rating of 0 represented a low grade or simply indicated a missing rating. Ther efore, these instances were not suitable for my analysis. Additionally, I removed irrelevant columns, focusing only on the essential data, such as rati ngs, book titles, and user IDs.

The dataset contained duplicate entries where multiple users had rated the same book more than once. Although these duplicates could have prov ided insights into user behaviour over time, I didn't have additional information such as time stamps between ratings. Without such contextual det ails, I decided to remove both ratings from each user.

Despite these data cleaning steps, the dataset still retained a substantial number of unique users and books for analysis. After cleaning and process ing, I was left with 46,623 unique book IDs, 43,132 unique user IDs, and 168,968 ratings (ranging from 1 to 10). It was also interesting to explore th e distribution of ratings, which revealed that the majority of ratings were higher than 6. The highest number of ratings fell in the range of 7-8, whic h was the case for 19,519 books. In order to use the book titles as integers for the model algorithm, I encoded them using the factorize function in pandas.

Number of books with average rating between 0 and 1: 126
Number of books with average rating between 1 and 2: 296
Number of books with average rating between 2 and 3: 586
Number of books with average rating between 3 and 4: 1135
Number of books with average rating between 4 and 5: 5428
Number of books with average rating between 5 and 6: 8570
Number of books with average rating between 7 and 8: 19519
Number of books with average rating between 8 and 9: 15883
Number of books with average rating between 9 and 10: 11057

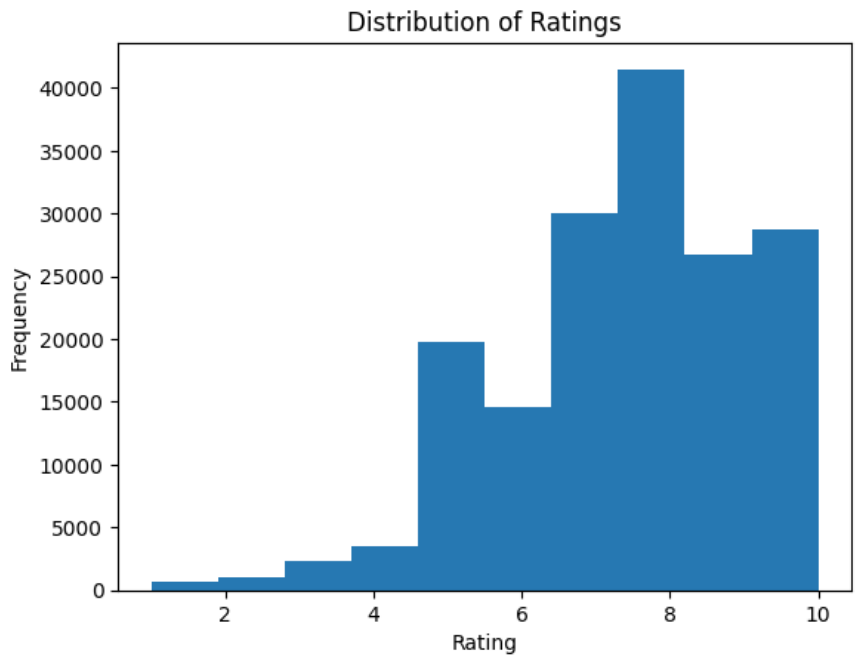


Figure 1. The histogram plot shows the distribution of ratings across all books in the dataset

Machine Learning Approach.

Following the data cleaning process, several models were considered for further analysis. Initial attempts involved experimenting with different frameworks, such as PyTorch, which were previously introduced in the class. However, the outcomes were not satisfactory in terms of performance. Therefore, considering time constraints, I decided to explore a Keras model, which was also discussed in class.

In my initial attempt, I employed a collaborative filtering model that utilized embeddings to incorporate user and book information for predicting ratings. However, upon analysing the plot (Model 1, Figure 2), it became apparent that the validation loss exhibited a significantly higher value compared to the training loss. This observation strongly indicated the presence of overfitting, whereby the model struggled to generalize effectively to unseen data. To address this issue, I proceeded to experiment with different parameters aiming to improve the model's performance.

It is important to note that due to computational limitations, the number of epochs had to be reduced.

Model 1.

```
# code from https://keras.io/examples/structured_data/collaborative_filtering_movielens/

EMBEDDING_SIZE = 50


class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_books, embedding_size, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_books = num_books
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.book_embedding = layers.Embedding(
            num_books,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.book_bias = layers.Embedding(num_books, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        book_vector = self.book_embedding(inputs[:, 1])
        book_bias = self.book_bias(inputs[:, 1])
        dot_user_book = tf.tensordot(user_vector, book_vector, 2)
        # Add all the components (including bias)
        x = dot_user_book + user_bias + book_bias
        # The sigmoid activation forces the rating to between 0 and 1
        return tf.nn.sigmoid(x)


model = RecommenderNet(num_users, num_books, EMBEDDING_SIZE)
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(), optimizer=keras.optimizers.Adam(lr=0.001)
)
```

```
Epoch 1/5
4780/4780 [==============================] - 626s 130ms/step - loss: 0.5998 - val_loss: 0.5715
Epoch 2/5
4780/4780 [==============================] - 768s 161ms/step - loss: 0.5673 - val_loss: 0.5871
Epoch 3/5
4780/4780 [==============================] - 625s 131ms/step - loss: 0.5725 - val_loss: 0.5900
Epoch 4/5
4780/4780 [==============================] - 531s 111ms/step - loss: 0.5827 - val_loss: 0.6035
Epoch 5/5
4780/4780 [==============================] - 528s 110ms/step - loss: 0.5967 - val_loss: 0.6042
```

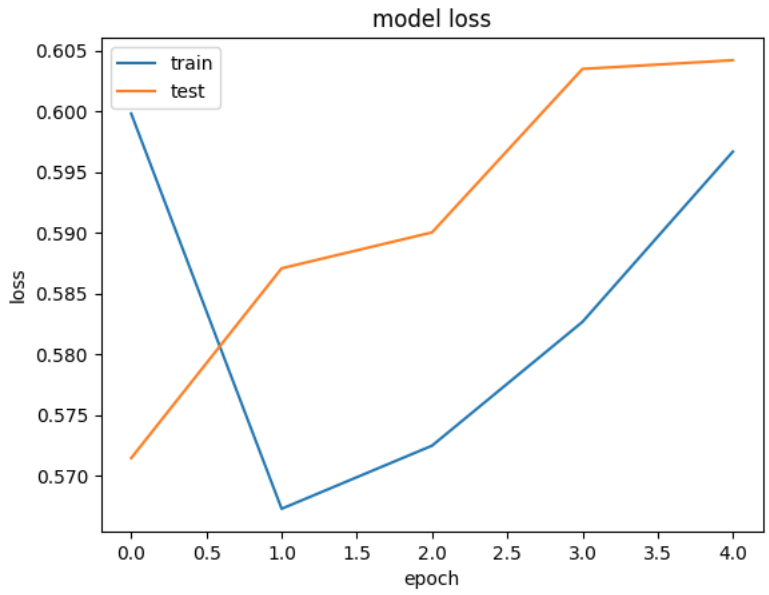


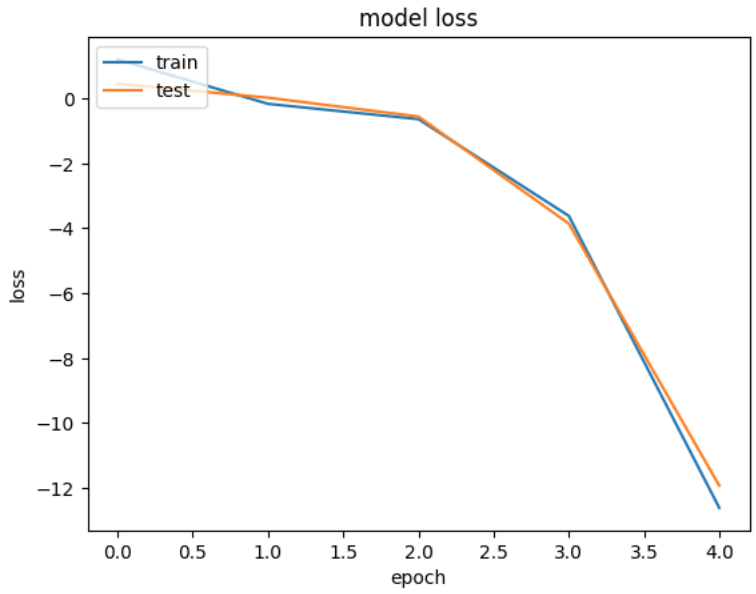
Figure 2. Model 1. Training and validation loss values recorded

Below, you can observe a selection of plots that represent the outcomes of my experimentation (Figure 2, Figure 3). Due to limitations in word count, I am not expanding on the technical details of the model. Nevertheless, it is noteworthy that in Model 2 (Figure 2), the modifications made, including the introduction of a regularization_ weight parameter and an adjustment of the embedding size to 60, resulted in negative loss values during the training process.

Model 2.

```python
# Code from https://keras.io/examples/structured_data/collaborative_filtering_movielens/

from tensorflow.keras.losses import MeanSquaredError, MeanAbsolutePercentageError


EMBEDDING_SIZE = 60


class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_books, embedding_size, regularization_weight, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_books = num_books
        self.embedding_size = embedding_size
        self.regularization_weight = regularization_weight
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(self.regularization_weight),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.book_embedding = layers.Embedding(
            num_books,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(self.regularization_weight),
        )
        self.book_bias = layers.Embedding(num_books, 1)



    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        book_vector = self.book_embedding(inputs[:, 1])
        book_bias = self.book_bias(inputs[:, 1])
        dot_user_book = tf.tensordot(user_vector, book_vector, 2)
        # Add all the components (including bias)
        x = dot_user_book + user_bias + book_bias
        # The sigmoid activation forces the rating to between 0 and 1
        return tf.nn.sigmoid(x)


model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=keras.optimizers.Adam(learning_rate=0.001))
```

```
Epoch 1/5
2097/2097 [==============================] - 179s 84ms/step - loss: 1.2008 - val_loss: 0.4530
Epoch 2/5
2097/2097 [==============================] - 158s 75ms/step - loss: -0.1660 - val_loss: 0.0234
Epoch 3/5
2097/2097 [==============================] - 184s 88ms/step - loss: -0.6367 - val_loss: -0.5573
Epoch 4/5
2097/2097 [==============================] - 161s 77ms/step - loss: -3.6179 - val_loss: -3.8560
Epoch 5/5
2097/2097 [==============================] - 153s 73ms/step - loss: -12.6080 - val_loss: -11.9261
```




```python
# Evaluate the model on the test data. Code from https://www.projectpro.io/recipes/make-predictions-keras-model#mcetoc_1g21q2u52e
score = model.evaluate(x=x_val, y=y_val, verbose=0)
print('Test loss:', score)
```

```
Test loss: -28.42108917236328
```

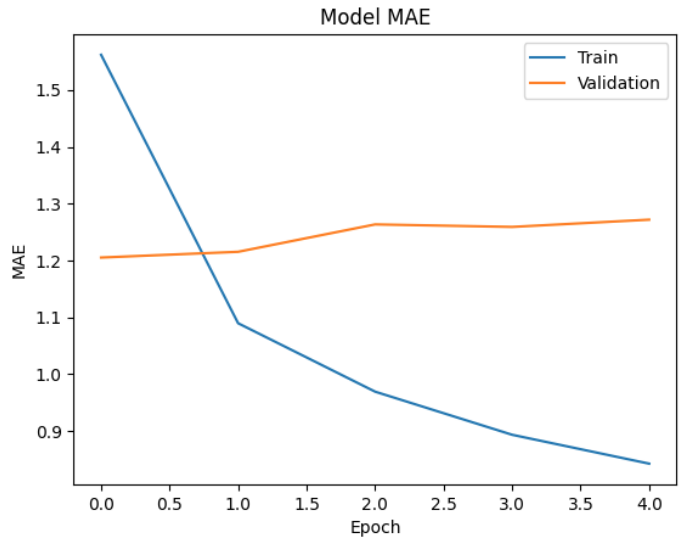
Figure 3. Model 2. Regularization_Weight Parameter

Considering that this issue could be attributed to the choice of the loss function, I made revisions to the model by using the Mean Squared Error (MSE) loss function (Figure 4, Model 3). Below, I have included the training code for that model, which also incorporates the experimentation of a concatenate layer operation. The training history indicates an improvement, as the model's MSE and Mean Absolute Error (MAE) values decrease during the training process, indicating that the model is learning and improving with each epoch. However, it is worth noting that the validation loss and validation MAE values show an increasing trend, suggesting that the model may be overfitting to the training data and not generalizing well to unseen data.

Model 3.

```
embedding_size = 50
epochs = 5

user = tf.keras.Input(shape=(1,))
book = tf.keras.Input(shape=(1,))

user_embedding = layers.Embedding(num_users, embedding_size, embeddings_regularizer=keras.regularizers.l2(1e-5))(user)
user_embedding = layers.Reshape(target_shape=(embedding_size,))(user_embedding)
book_embedding = layers.Embedding(num_books, embedding_size, embeddings_regularizer=keras.regularizers.l2(1e-5))(book)
book_embedding = layers.Reshape(target_shape=(embedding_size,))(book_embedding)

concat = layers.Concatenate()([user_embedding, book_embedding])
dense1 = layers.Dense(256, activation="relu")(concat)
dropout = layers.Dropout(0.2)(dense1)
output = layers.Dense(1)(dropout)

model = tf.keras.Model(inputs=[user, book], outputs=output)
model.compile(loss=tf.keras.losses.MeanSquaredError(), optimizer=tf.keras.optimizers.Adam(lr=0.001), metrics=["mae"])

history = model.fit(x=[x_train[:, 0], x_train[:, 1]], y=y_train, batch_size=64, epochs=epochs, verbose=1, validation_data=([x_val[:, 0], x_val[:, 1]], y_val))

plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('Model MAE')
plt.ylabel('MAE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
Epoch 1/5
1864/1864 [==============================] - 133s 70ms/step - loss: 5.0034 - mae: 1.5618 - val_loss: 2.5007 - val_mae: 1.2051
Epoch 2/5
1864/1864 [==============================] - 119s 64ms/step - loss: 2.1298 - mae: 1.0895 - val_loss: 2.5914 - val_mae: 1.2152
Epoch 3/5
1864/1864 [==============================] - 120s 64ms/step - loss: 1.7496 - mae: 0.9691 - val_loss: 2.7612 - val_mae: 1.2634
Epoch 4/5
1864/1864 [==============================] - 118s 63ms/step - loss: 1.5339 - mae: 0.8933 - val_loss: 2.8004 - val_mae: 1.2590
Epoch 5/5
1864/1864 [==============================] - 117s 63ms/step - loss: 1.3914 - mae: 0.8425 - val_loss: 2.8951 - val_mae: 1.2718
```

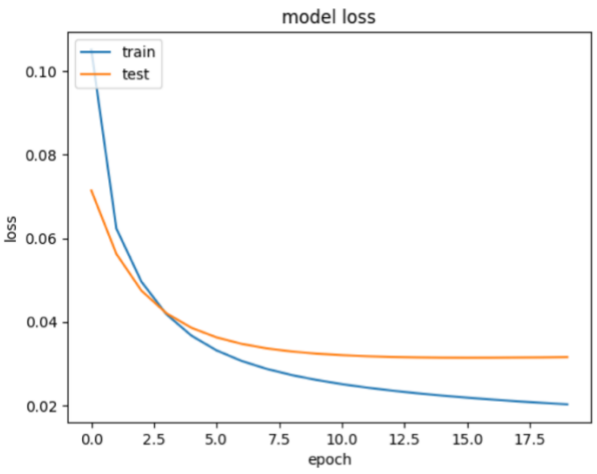



```
def catalog_coverage(recommended_items, catalog_items):
    unique_recommended_items = set(recommended_items.flatten())
    unique_catalog_items = set(catalog_items.flatten())

    coverage_score = len(unique_recommended_items) / len(unique_catalog_items)
    return coverage_score


recommended_items = model.predict([x_val[:, 0], x_val[:, 1]])  # Recommended item predictions from the model
catalog_items = df_subset["book"].unique()  # All available book IDs in the catalog

coverage_score = catalog_coverage(recommended_items, catalog_items)
print("Catalog Coverage: {:.2%}".format(coverage_score))
```

```
932/932 [==============================] - 1s 2ms/step
Catalog Coverage: 30.56%
```

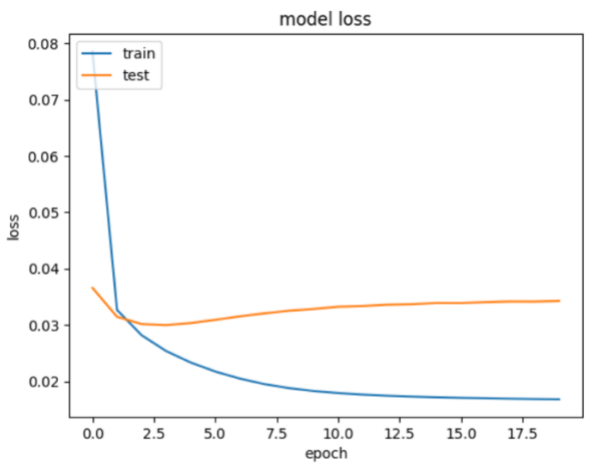
Figure 4. Model 3. (MSE) loss function

I continued experimenting with the architecture of the model and developed a model (Model 4) that I believe achieved good recommendation levels. Using this model's architecture as a base, I experimented with different characteristics and developed variations of the model (Figure 5). The code for each of these models can be found in the provided notebooks.



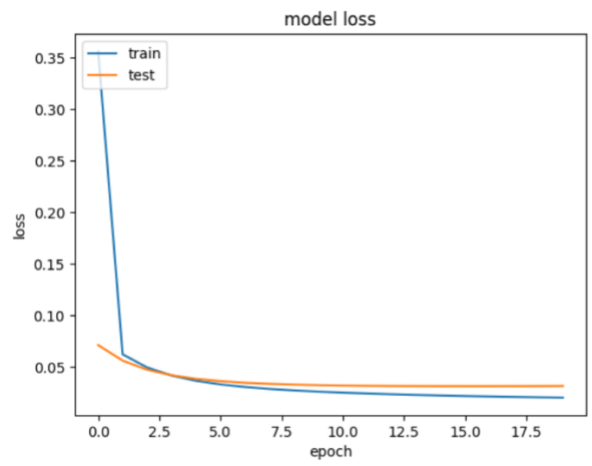
Model 4:

Embedding size: 60
Regularization weight: 0.01
Training data percentage: 90%
Optimizer: Adam with learning rate 0.001
Loss function: MeanSquaredError



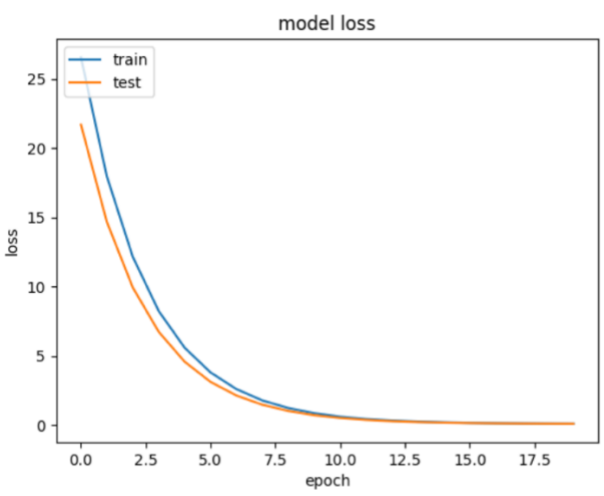
Model 5

Embedding size: 60
Regularization weight: 0.01
Training data percentage: 90%
Additional layer: Dense layer with 32 units and ReLU activation
Optimizer: Adam with learning rate 0.001
Loss function: MeanSquaredError



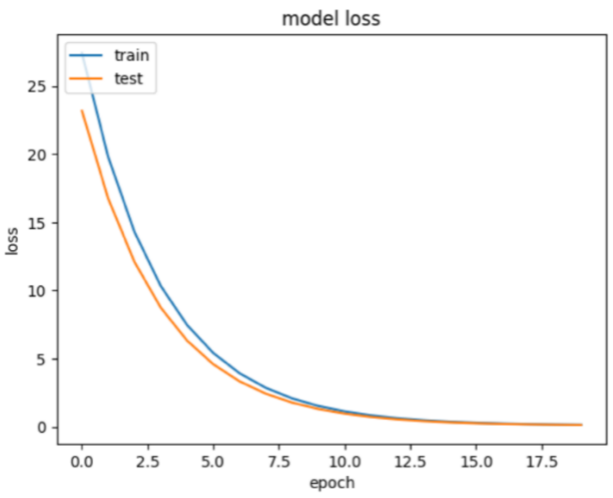
Model 6

Embedding size: 80
Regularization weight: 0.1
Training data percentage: 90%
Optimizer: Adam with learning rate 0.001
Loss function: MeanSquaredError



Model 7

Embedding size: 80
Regularization weight: 0.1
Training data percentage: 90%
Optimizer: SGD with learning rate 0.001
Loss function: MeanSquaredError



Model 8

Embedding size: 80
Regularization weight: 0.1
Training data percentage: 75%
Optimizer: SGD with learning rate 0.001
Loss function: MeanSquaredError

Figure 5.

For Model 4, the data was pre-processed by shuffling the dataset and splitting it into training and validation sets, with a 90% training and 10% validation split. The model architecture consists of three embedding layers: one for user embeddings, one for book embeddings, and additional embedding layers for user bias and book bias. The user and book embeddings are initialized with the "he_normal" initializer and regularized using L2 regularization with a regularization weight of 0.01 to control the model's complexity and prevent overfitting. The embeddings for users and books have a size of 60, meaning each user and book is represented by a 60-dimensional vector. The inputs, user embeddings, book embeddings, user bias, and book bias are multiplied element-wise and summed together to obtain the dot product of the user-book interaction. This result, along with the user bias and book bias, is passed through a sigmoid activation function to produce the final rating prediction. Mean squared error (MSE) is used as the loss function to measure the difference between the predicted ratings and the actual ratings. Adam optimizer is used with a learning rate of 0.001 to update the model's parameters during training.

On evaluating this model and the rest, a composition of evaluation metrics was used, including F1 score, Catalog coverage, Average Diversity, Test loss (Test score), and Mean Average Precision (MAP @10).

Model 4 demonstrated good performance overall, with a relatively low test loss indicating a good fit between the predicted and actual ratings. While the catalog coverage of 27.94% suggests that the model covers approximately one-fourth of the total items available in the catalog, providing a decent range of recommendations. The F1 score reflects the proportion of relevant items among the recommended items, indicating a good level of relevance. However, the MAP @10 score for Model 4 was significantly higher at 0.9956 compared to Model 5 with a score of 0.1357. A higher MAP @10 score suggests better accuracy and relevance of the top-10 recommended items.

Reflecting on these metrics, the decision was made to add a new layer to the existing architecture (Model 5) to potentially improve the performance of the recommendation system. The addition of a new layer introduces more parameters and complexity to the model, enabling it to capture and learn more intricate patterns and relationships in the data. This enhancement could improve the model's ability to understand and capture the underlying dynamics of user-item interactions, potentially leading to more accurate and personalized recommendations.

In Model 5, although the test loss did not decrease compared to Model 4, the catalog coverage increased, indicating a larger portion of the item catalog being covered by the recommendations. The higher F1 score suggests a greater proportion of recommended items being truly relevant to the user's preferences. However, it is important to note that Model 4 achieved a significantly higher MAP @10 score of 0.9956 compared to Model 5, which scored 0.1357. The higher MAP @10 score in Model 4 suggests better accuracy and relevance in the top-10 recommended items.

Model	Test Loss	Average Diversity	Catalog Coverage	MAP @10	F1-score
4	*0.0315*	*0.9975*	*27.94%*	*0.9956*	*0.8939*
5	*0.0343*	*0.9829*	*53.91%*	*0.1357*	*0.9173*
6	*0.0317*	*0.9994*	*27.94%*	*1.0000*	*0.8938*
7	*0.1094*	*1.0000*	*27.94%*	*1.0000*	*0.6933*
8	*0.1432*	*1.0000*	*69.02%*	*1.0000*	*0.6933*

Figure 6. Evaluation Metrics for Models.

Working on the previous model's architecture (Model 4), I experimented with the embedding size and regularization weight to capture more nuanced features in Model 6. However, the changes resulted in quite similar metric values. To further improve the model's performance, I decided to switch the optimizer from Adam to SGD (Stochastic Gradient Descent) in Model 7. Comparing the evaluation metrics, Model 4 generally outperformed Model 7 in terms of test loss and F1-score, while Model 7 performed slightly better in terms of average diversity and MAP @10.

I then kept the same architecture for Model 8 but changed the percentage of training data from 75% to 90%. Model 8 showed better performance in terms of average diversity, MAP @10, and catalog coverage, with a significantly higher percentage of 69.02%.

Model Comparison

When comparing the models (Figure 7), it is evident that there isn't a single model that holistically outperforms the rest, with all providing good level of accuracy and fairness. The choice of the model depends on the specific requirements and objectives of the recommendation system.

For example,Model 8, with a higher percentage of training data (90%), may be beneficial when the goal is to cover a larger portion of the catalog and provide recommendations from a wide range of items. This ensures better coverage and exposure of different products to users. Additionally, Model 8 achieves a high MAP @10 score of 1.0000, indicating accurate ranking of the top recommendations. If maximizing catalog coverage and providing accurate top recommendations are important factors, Model 8 would be a suitable choice.

On the other hand, if precision and overall accuracy are the primary concerns, Model 4 stands out. It demonstrates a relatively low test loss of 0.0315 and a high F1-score of 0.8939. These metrics suggest good predictive accuracy and a balanced performance between precision and recall. If the priority is accurate recommendations without sacrificing relevance, Model 4 would be a strong contender.

Ultimately, the selection of the best model depends on the specific goals and requirements of the recommendation system. It is crucial to consider factors such as catalog coverage, accuracy, precision, and relevance when making the decision.

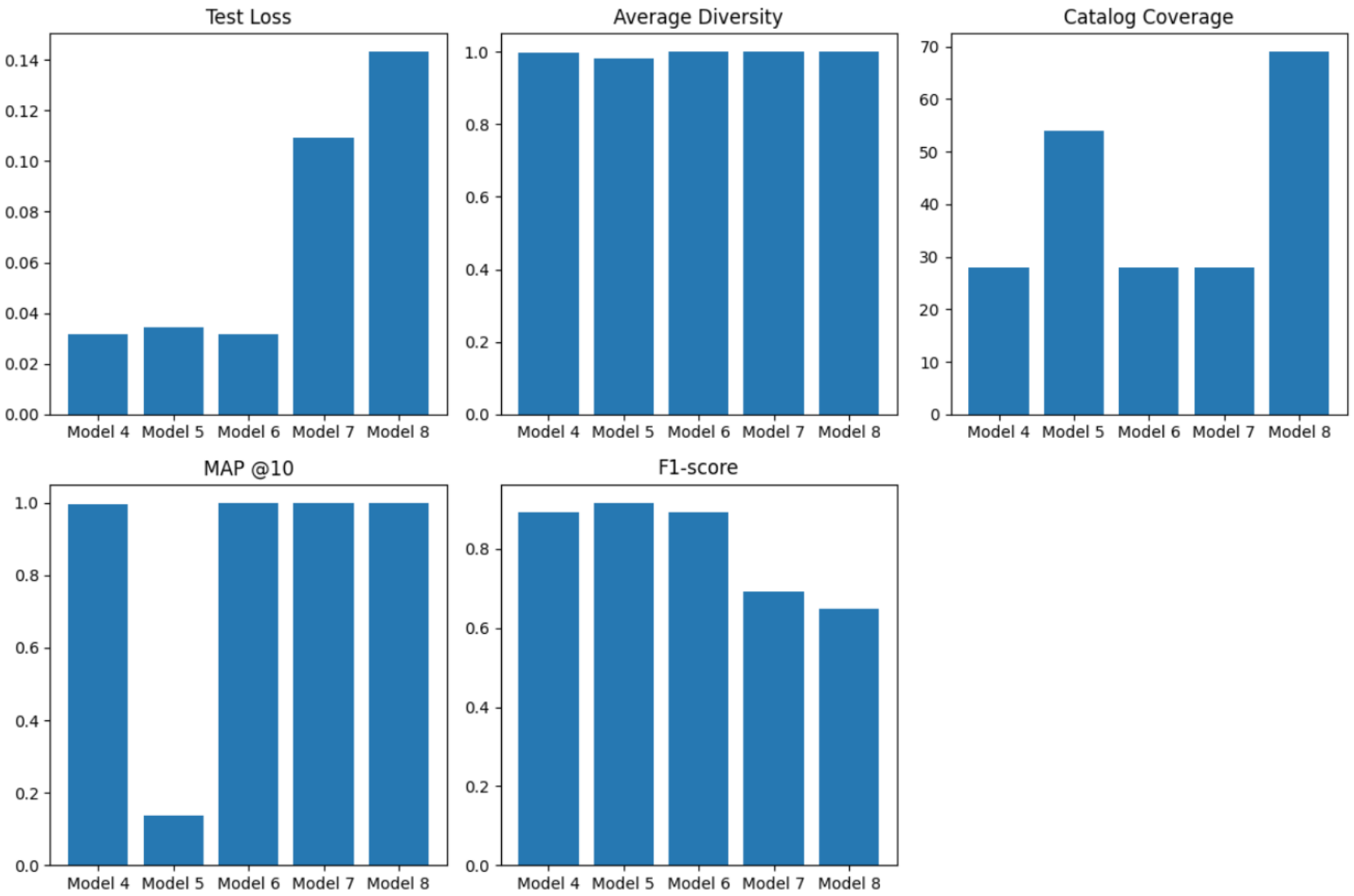


Figure 7.

Reflections.

Overall, this project provided me with an opportunity to apply and expand upon the skills I acquired in previous units. I was able to utilise and develop my technical skills in coding, sourcing and adapting existing code, as well as gain a deeper appreciation for the various ways in which data can be processed, cleaned, and visualized. The project's primary objective was to build and develop a machine learning model, which initially started with non-functional code and required extensive debugging. Throughout the project, I learned through research and experimentation, resulting in a significantly improved and accurate functioning model with various architecture characteristics.

Given more time, there are a few areas where I could have further enhanced the model. One possible improvement would have been to incorporate additional side information about users and items into the model. This could include features such as user demographics and age, which could have potentially improved the model's ability to capture complex user-item interactions and generate more accurate recommendations. Furthermore, I would have explored the architecture of the model in more detail. This could have involved experimenting with different layers, activation functions, or regularization techniques to further improve the model's performance and generalization capabilities. Additionally, I could have incorporated Data Augmentation techniques to increase the size and diversity of the training data. This would have helped the model learn more robust and representative patterns from the data, potentially leading to better recommendations.

In summary, while this project allowed me to develop and deploy a successful machine learning model, there are always areas for further exploration and improvement. By incorporating additional side information, exploring different architectural variations, and utilizing data augmentation techniques, the model's performance and accuracy could have been further enhanced.

The code for this project is available in the provided notebooks. I primarily used code snippets from the weekly notebooks of this and previous terms, and also referred to code from the Coding 3 notebooks. Whenever I utilized code from external sources, I made sure to include proper comments indicating the source.
Throughout the project, I relied on Chat GPT to proofread and improve the document. It assisted me in debugging any errors and organizing the code in a more structured manner.

Reference:
RUCHI. (2023). Book Recommendation Dataset. [Online]. Available at: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset (Accessed: 6th June 2023).

Have a great Summer! And thank you! ☺☺☺