COMP4332 Project 3 Report

Group 28

HUI Man Wah, WONG Ho Leong, WONG Yu Ning, YIP Nga Yin

Introduction

This project aims to perform rating prediction for the products given their reviewer ids and product ids. This report details the approaches and techniques, specifically **Collaborative Filtering based Recommendation** and **Wide & Deep Learning**, employed to build a model for this rating prediction task.

Dataset Description

There are 52697, 6596 and 6597 data in the training set, validation and testing set respectively. The training dataset includes 4 feature columns, namely the *ReviewerID*, *ProductID*, Text, Summary. For validation and testing dataset, only the *ReviewerID* and *ProductID* are provided.

Additionally, we are provided with a products profile. There are a total of 16+13 columns in the product profile.

Due to time limitation, we are unable to utilize the Text and Summary column presented in the training set. In this report, we will only focus on using *ReviewerID*, *ProductID* and the Product profile for rating prediction.

Exploratory Analysis:

Product data Analysis:

In the product profile, excluding the details column, there are a total of 16 columns. In details columns, there are 13 unique columns. Notably, some are empty columns or do not exist for certain products. It would be necessary for us to clean the product profile before using it for prediction. Below summarizes the column information in the product profile.

All the column names are:

category	tech1	description	fit	title
tech2	brand	feature	rank	main_cat
similar_item	date	price	imageURL	imageURLHighRes
ProductID				
Publication Date	Simultaneous Device Usage	Publisher	Release Date	Language
Print Length	File Size	ISBN-10	ISBN-13	Page Numbers Source ISBN
ASIN	Word Wise	Lending		

The empty columns are shown below:

tech1	description	fit	tech2	feature
similar_item	date	imageURL	imageURLHighRes	

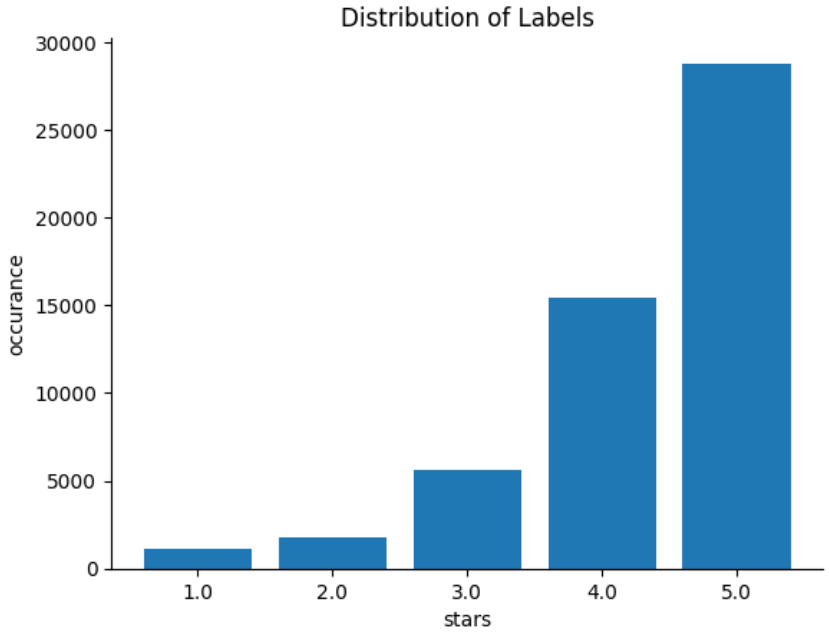
Columns that do not exist for all products in the profile:

Word Wise	Publication Date	ISBN-13	Lending
Publisher	Simultaneous Device Usage	Page Numbers Source ISBN	Release Date
Print Length	ISBN-10		

The empty columns would be removed in the data preprocessing section. For missing values, we would assign an “unknown” or “0”. Additionally, some columns like "Rank" store numerical data but in string format. Extracting the numerical information from these columns and convert them to numerical features might help to improve the model's predictive accuracy.

Training dataset Analysis:

Class imbalance problem is serious in the training dataset. As we can see from the graph below, class 1 and 2 have particularly small amount of training data. Data resampling may require boosting the accuracy of the model.



Model Training:

Due to the different data requirements of the Collaborative Filtering (CF)-based recommendation model and the Wide and Deep model, we employed distinct preprocessing techniques for each approach. These preprocessing steps will be briefly discussed in the corresponding sections.

<u>CF-based recommendation Model:</u>

Data Resampling:

As mentioned in the Training Dataset Analysis Section, we observed a serious class imbalance issue. To mitigate this issue, we tried to implement the Oversampling with SMOTE and Under sampling with class weights, The results are shown below:

Techniques Used	***RMSE***
Baseline model	0.8496
Oversampling using SMOTE	1.4435
Undersampling using Class weights	1.3802

As we can see from the table above, oversampling and under sampling gives significantly higher RMSE. It is possible that CF-based model may be too simple to

benefit from the data resampling. As a result, we did not use any data resampling techniques in our CF-based model training process.

Grid Search:

To look for the optimal parameters in a more efficient manner, we only train each model with 1 epoch, and set the dropout rate search space with larger offset. Below summarized the range of model parameters.

```
embedded size = [8,16,32,64,128,256]
hidden_units = Combination([512,256,128,64,32,16])
dropouts = np.arange(0,1,0.2)
```

For the hidden layer, the search space included all combinations of 1, 2, and 3 hidden layers. Each with hidden unit counts selected from the above range, and paired with dropout layers to form a dense -> dropout [-> dense -> dropout] [-> dense -> dropout] structure.

The best validation RMSE and the corresponding model parameters we found are shown below:

```
[ ] best_val_rmse = min(model_params_and_validationloss.values())
    min_key = [key for key, value in model_params_and_validationloss.items() if value == best_val_rmse]
    print("best validation loss: ",best_val_rmse, "model params: ",min_key[0])

best validation loss: 0.8421806456970286 model params: (16, (128, 64), 0.0)
```

We can see that the model with embedded size of 16, two hidden units (128,64), and dropout rate of 0 has already passed the competitive baseline and obtain a RMSE of 0.8422.

To further optimize the model, we finetune the dropout parameters with the following range:

```
dropouts = np.arange(0,1,0.1)
```

The code snippet below shows the best validation RMSE and the corresponding model parameters.

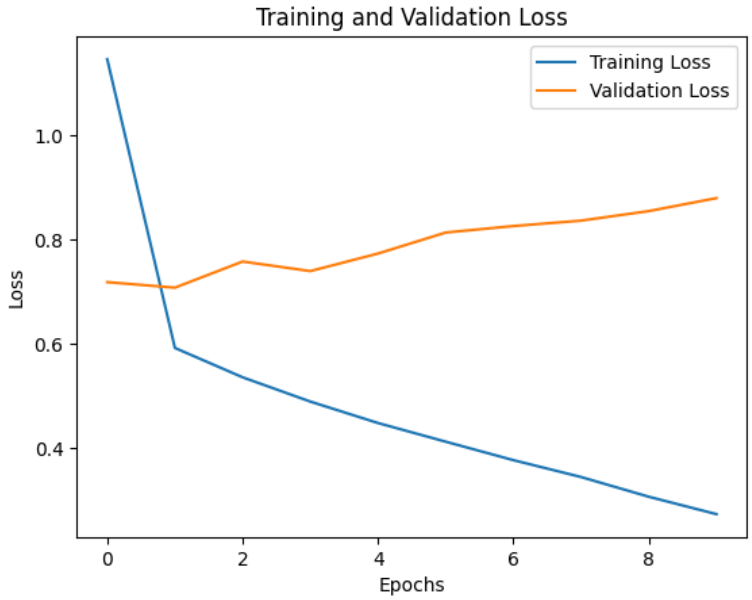
```
[ ] best_val_rmse = min(dropout_valLoss.values())
    min_key[0] = [key for key, value in dropout_valLoss.items() if value == best_val_rmse]
    print("best validation loss: ",best_val_rmse, "model params: ",min_key[0])

best validation loss: 0.8511009952494945 model params: [(16, (128, 64), 0.1)]
```

The slight increase in validation RMSE after fine-tuning the dropout rate is likely due

to the inherent randomness in the model's training process, where small variations in the weight initialization or training data can lead to marginal differences in performance metrics.

Look for optimal number of epochs:

Finally, we tried to train the model with 10 epoch and save the best model according to the validation loss. Below shows the training and validation loss as the number of epochs increases.



The model was observed to have quickly overfitted to the training data after a single epoch. As a result, the model that was trained for one epoch was saved, and the validation RMSE is shown below:

```
[ ]  #dim, hidden_layer, dropout = min_key[0][0]
     model = tf.keras.models.load_model('bestmodel.h5')#model name ha
     y_pred = model.predict([val_users, val_items])
     print("Validation set RMSE: ", rmse(y_pred, val_ratings))
     y_pred = model.predict([te_users, te_items])
     test_df["Star"] = np.round(y_pred).astype(int)
     #save the prediction
     test_df.to_csv("prediction.csv", index=False)

     207/207 [==============================] - 0s 572us/step
     Validation set RMSE: 0.8415865899597444
     207/207 [==============================] - 0s 559us/step
```

Wide & Deep Learning Model:

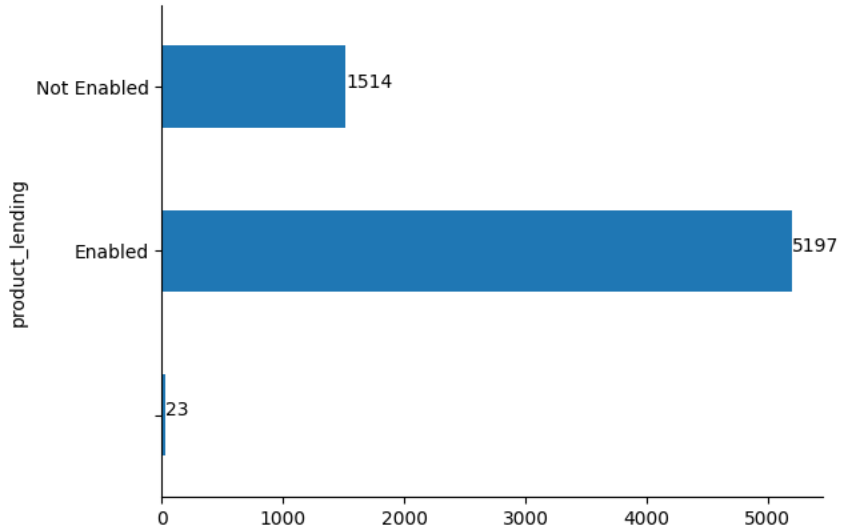
Data processing:

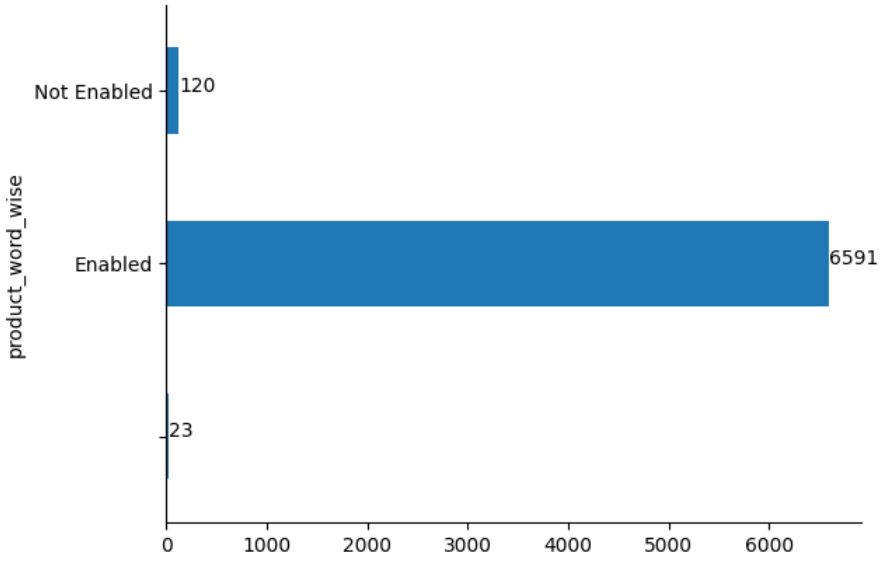
- Extract Product Information:

  Certain columns, such as '*rank*', '*File Size*', and '*Print Length*', contained numeric values that were stored as strings. These columns would be converted from string to numeric data types to enable proper numerical analysis.

- Missing Product Information:

  Additionally, we have identified instances where essential data fields in the product.json file are incomplete or missing. The following shows two examples: *Lending* and *Word Wise* columns.





As we can see both columns contain 23 missing values. We handled these missing values by assigning the value “unknown”.

- Unknown Product Information:

  The product profile did not include every item in the training set, with 25 *ProductIDs* unknown. To avoid errors, a default product profile was assigned for these missing *ProductIDs* during training and testing, ensuring a consistent and reliable analysis despite the data limitation.

```
default_values = {
        'product_category': [],
        'product_title': "unknown",
        'product_brand': "unknown",
        'product_rank': 0,
        'product_main_cat': "unknown",
        'product_price': 'unknown',
        'ProductID': '',
        'product_file_size': 0,
        'product_print_length': 0,
        'product_publisher': "unknown",
        'product_word_wise': "unknown",
        'product_lending': "unknown",
        'product_language': "unknown"
}
```

Model Implementation:

The model architecture has two main components: **Wide Model** and **Deep Model**.

The Wide Model processes the wide features and employs linear regression to capture the co-occurrences of these crossed features.

Meanwhile, the Deep Model handles the deep features through a deep neural network with fully connected layer. This network is designed to capture complex, non-linear relationships between features. It includes fully connected layers, each followed by ReLU activation functions and dropout layers to mitigate overfitting.

The outputs of the Deep Model are then combined with the wide features and fed into a single fully connected layer to predict the star rating a user will give to an item.

The following are the columns we used as wide and deep features respectively:

Deep Continuous features:

product_rank	product_file_size	product_print_length
product_word_wise	product_lending	product_language
product_title	product_brand	product_publisher

Wide features:

The Category column has 52 unique values. We extracted two-way, three-way, and four-way combinations of the categories as wide features, using only those within 1.5 standard deviations of the mean occurrence.

For the remaining column data, since only a small number of the product IDs have values, we decided to ignore it.

Grid Search:

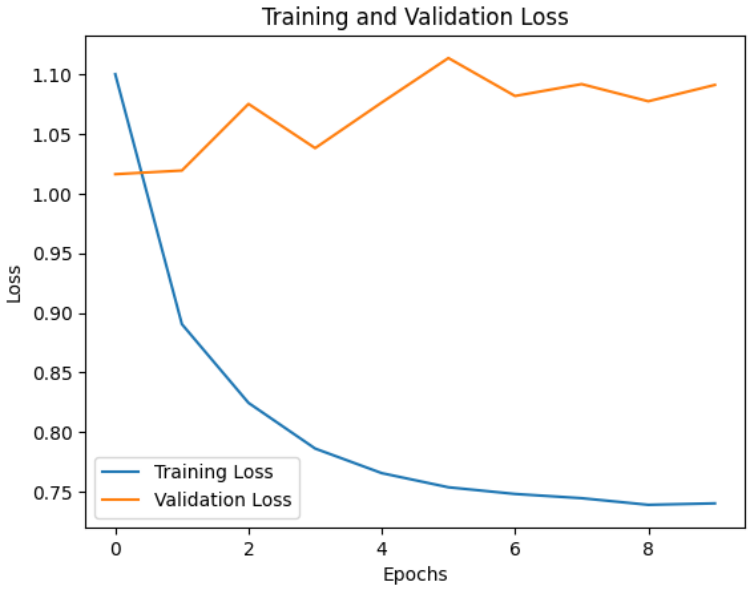
To look for the optimal parameters in a more efficient manner, we only train each model with 1 epoch, and set the dropout rate search space with larger offset. Below summarized the range of model parameters.

```
embedded size = [8,16,32,64,128,256]
hidden_units = Combination([512,256,128,64,32,16])
dropouts = np.arange(0,1,0.1)
```

For the hidden layer, the search space included all combinations of 1, 2, and 3 hidden layers. Each with hidden unit counts selected from the above range, and paired with dropout layers to form a dense -> dropout structure.

After searching, the best validation RMSE and the corresponding model parameters we found are shown below:

```
best_val_rmse = min(model_params_and_validationloss.values())
min_key = [key for key, value in model_params_and_validationloss.items() if value == best_val_rmse]
print("best validation loss: ",best_val_rmse, "model params: ",min_key[0])
[197] ✓ 0.0s
... best validation loss:  1.0114345378069607 model params:  (256, (512, 256), 0.5)
```



Similar to the CF-based model, the model quickly overfit after one epoch, hence we saved the model that is trained with only one epoch. The final validation RMSE is 1.0114.

Conclusion:

In conclusion, our experiments explored modern rating prediction techniques, specifically Collaborative Filtering-based Recommendation and Wide & Deep Learning. We discovered that CF-based recommendation model gives a better result compared to the wide and deep model. We achieved a validation RMSE of 0.8415.