School of Data Science
香港城市大學
City University of Hong Kong

SDSC3002 Data Mining
Final Project

School of Data Science

Model-based and Deep Learning-based Collaborative Filtering For Restaurant Recommendation System

Authors:

ABDINEGARA Beatrice (57103988)	babdinega2-c@my.cityu.edu.hk
GUSTAF Benedict Ronaldo (56680002)	rbgustaf2-c@my.cityu.edu.hk
HA Quang Minh (57040174)	quangmha2-c@my.cityu.edu.hk
YENNOTO Keane Dylan (56700945)	dkyennoto2-c@my.cityu.edu.hk

Community ♣

2022/2023 – Semester B

Work Distributions

Student Name	Student ID	Work Descriptions
ABDINEGARA Beatrice	57103988	Data Preprocessing
		Deep Learning Modeling
		PPT Editing
		Oral Presentation
		Report Writing
GUSTAF Benedict Ronaldo	56680002	Data Preprocessing
		Deep Learning Modeling
		PPT Editing
		Oral Presentation
		Report Writing
HA Quang Minh	57040174	Data Preprocessing
		Matrix Factorization Modeling
		PPT Editing
		Oral Presentation
		Report Writing
YENNOTO Keane Dylan	56700945	Data Preprocessing
		Matrix Factorization Modeling
		PPT Editing
		Oral Presentation
		Report Writing

Contents

1 Introduction

The advancement of the internet over the past ten years has significantly altered how we communicate, shop for goods, and even find a restaurant to eat. A system that can help us to find the right items we want is necessary due to the enormous number of items offered online. Thus, recommendation systems are powerful tools that can provide item suggestions that may be useful for users according to their preferences, as well as information that is either implicitly or explicitly inferred from the user's behavior or a description of the item and its characteristics [11]. The technology of recommender tools uses advanced algorithms to find a way to relate this information, extract meaningful patterns and identify connections between users and items. This allows the systems to provide tailored recommendations that can cater to each user's unique preferences, thus enhancing their overall experience.

In general, there are two different types of recommender systems, content-based filtering recommendation and collaborative filtering recommendation. Content-based filtering tries to find similarities between items based on their features and uses this information to recommend items similar to what the user has previously liked or interacted with [8]. For example, content-based filtering would recommend restaurants based on their features, such as ambience, restaurant categories, and cuisine types. It would focus on finding restaurants that are similar to the ones that a user has previously liked or visited before. Specifically, if a user has shown a preference for casual dining and outdoor seating, then a content-based system may recommend other restaurants that share similar features.

The other type of recommender system is collaborative filtering, which is the main strategy of recommender systems used for this project. In contrast, a collaborative filtering recommendation focuses on identifying patterns and similarities between users rather than attributes between items [2]. Hence, instead of recommending restaurants that are similar to the ones a user has liked in the past, this approach recommends restaurants based on the behavior of other similar users who have liked them in the past. The similarity of the users is computed based on their behavior, such as their past restaurant ratings. A set of "nearest neighbor" users is identified with the strongest correlation to the target user's past ratings. Finally, scores for unseen restaurants are predicted based on a combination of the scores known from the nearest neighbors. This collaborative approach differs from content-based filtering, which relies on information about the features of the restaurants to make recommendations.

In general, there are two major approaches to collaborative filtering: memory-based and model-based. Memory-based collaborative filtering involves analyzing the entire user database to make predictions. This method works well with smaller datasets and is frequently easier to use. However, as the dataset grows larger, scalability may become a problem. Model-based collaborative filtering, in contrast, makes predictions by estimating or learning a model from the user database [10]. This method is frequently more efficient for larger datasets and has the potential to be more accurate than memory-based ones. The interaction matrix between items and users is modeled via model-based collaborative filtering. This model maps the characteristics of each user and item to a specific embedding space. The embeddings are then learned through a machine-learning process, allowing similar users and items to be mapped to embeddings that are closer together. The most commonly used model-based collaborative filtering is Matrix Factorization (MF), by learning latent factors to represent the characteristics of users and items, which then breaks down the user-item interaction matrix into the product of two smaller rectangular matrices with lower dimensions. As a result, the hidden factors that drive user preferences and item characteristics can be identified.

Another popular approach for recommendation systems is deep-learning-based collaborative filtering, which uses multiple layers of neural networks to learn the representations of users and items. Thus, high-quality low-dimensional embeddings can be created, which is important for predicting and recommending items that are close in the embedding space.

This project's main objective is to explore two different collaborative filtering strategies in building our recommendation system: matrix factorization and deep learning to predict users' ratings.

2 Data Preprocessing and Feature Engineering

We utilized Yelp Dataset for this project. Yelp Dataset is a dataset that contains business reviews developed by an American multinational company, Yelp, which operates an online platform and mobile app for users to search for businesses and write reviews. The dataset consists of businesses information, users information, collection of reviews, tips, check-ins, and photos, each in a separate JSON file. The users, tips, check-ins, and photos were not used in this project.

The `business.json` contains the business ID and name, the location (including the address, city, state, postal code, latitude, and longitude), the average star rating of this business, the review count, some attributes related to parking and takeout availability, business categories, and opening hours.

The `review.json` contains the text review together with the user ID of the author, the business ID for whom the review was made, the star rating, and the review date.

We began our preprocessing by loading the `yelp_academic_dataset_business.json` and `yelp_academic_dataset_review.json`. The JSON files are then transformed into a pandas dataframe. The dataframes will be referred to as `business_df` and `review_df`, respectively.

The Yelp dataset contains businesses of various kinds, not only restaurants. Therefore, we only keep rows within `business_df` with `restaurant` or `restaurants` in their categories attribute. Then, we merged the `business_df` and `review_df` files by the `business_id`. This merge resulted in a dataset that includes ratings, user id, business id, text reviews, and other relevant features of the restaurants.

After reviewing the merged data, we found that the dataset was free of null or identical values, indicating that it was clean. However, we found that some users review the same restaurant multiple times, which could lead to biased recommendations. Therefore, we decided only to keep the latest review given by each user for a specific restaurant. This filter helped ensure our reviews were up-to-date.

Next, we analyzed the data by grouping the dataset by cities. We decided to use restaurants located only in Philadelphia, the city with the most review count. By focusing on a specific city, we could provide more accurate recommendations for users interested in Philadelphia restaurants.

Following this, we decided only to include users who had provided ten or more reviews to ensure the reliability of our dataset. This allowed us to have a dataset with a sufficient number of reviews from each user. Through this filtering, we ensured that our models would be trained on a substantial amount of reviews after the data-splitting process, while also ensuring that there would be enough actual reviews remaining to be used for evaluation.

Finally, we decided to create a sentiment analysis score from text reviews. The sentiment analysis was done through a pre-trained model VADER or Valence Aware Dictionary and Sentiment Reasoner. While not being trained specifically on restaurant reviews, Vader is trained on Twitter tweets, movie reviews, technical product reviews, and opinion articles. Furthermore, Vader is able to capture word-order sensitive relationships between terms, such as punctuation, capitalization, and others [4]. Thus, we chose VADER as the model for our sentiment analysis. We only used the `compound_score` generated from the VADER model and associated it with each respective review. The `compound_score` is the aggregated version of the sentimental features within the text. Then, the VADER built-in package in NLTK normalized the values to the range $[-1, 1]$, using the function:

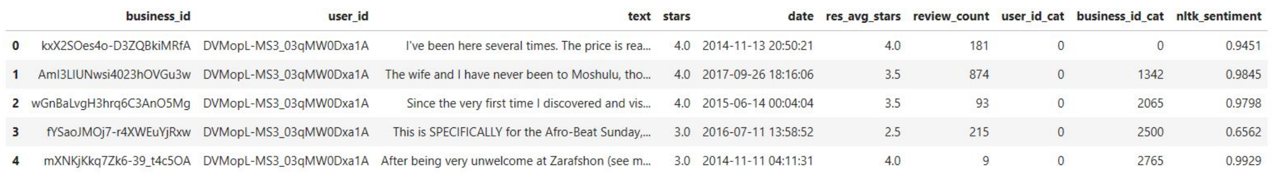
$$\hat{x} = \frac{x}{\sqrt{x^2 + \alpha}}$$

where α is set to be 15 which approximates the maximum expected value of x.

Overall, these preprocessing steps lead us to a clean dataset, which is relevant for our recommendation system. Table 1 shows the final composition of the dataset used and Figure 1 shows the final form of the dataset.

Number of Ratings	295003
Number of Businesses	10727
Number of Users	57951

Table 1: The final composition of dataset used in this project



	business_id	user_id	text	stars	date	res_avg_stars	review_count	user_id_cat	business_id_cat	nltk_sentiment
0	kxX2SOes4o-D3ZQBkiMRfA	DVMopL-MS3_03qMW0Dxa1A	I've been here several times. The price is rea...	4.0	2014-11-13 20:50:21	4.0	181	0	0	0.9451
1	AmI3LIUNwsi4023hOVGu3w	DVMopL-MS3_03qMW0Dxa1A	The wife and I have never been to Moshulu, tho...	4.0	2017-09-26 18:16:06	3.5	874	0	1342	0.9845
2	wGnBaLvgH3hrq6C3AnO5Mg	DVMopL-MS3_03qMW0Dxa1A	Since the very first time I discovered and vis...	4.0	2015-06-14 00:04:04	3.5	93	0	2065	0.9798
3	fYSaoJMOj7-r4XWEuYjRxw	DVMopL-MS3_03qMW0Dxa1A	This is SPECIFICALLY for the Afro-Beat Sunday,...	3.0	2016-07-11 13:58:52	2.5	215	0	2500	0.6562
4	mXNKjKkq7Zk6-39_t4c5OA	DVMopL-MS3_03qMW0Dxa1A	After being very unwelcome at Zarafshon (see m...	3.0	2014-11-11 04:11:31	4.0	9	0	2765	0.9929

Figure 1: First 5 rows of the final dataset

To take into account the evaluation of our recommendation system, we used simple cross-validation by splitting the dataset into approximately 20% of the ratings withheld as part of the test set. The ratings were selected across all known reviews ensuring a representative sample of the population. The rest of the data, approximately 80%, is used for the purpose of training.

3 Evaluation Metrics

The main metric we used to evaluate the performance of our recommendation system models is Root Mean Squared Error (RMSE). Our objective functions for both matrix factorization and deep learning models were squared loss functions, which are directly related to RMSE:

$$RMSE = \sqrt{\frac{1}{|S|} \sum_{(u,i) \in S} (\hat{r_{ui}} - r_{ui})^2}$$

where S is the set of ratings in the test set, r_{ui} and $\hat{r_{ui}}$ are the actual and predicted rating of user u for restaurant i respectively. In addition to RMSE, we also compare the Mean Absolute Error (MAE) of our models. MAE measures the absolute difference between the predicted and actual ratings, and is calculated as follows:

$$MAE = \frac{1}{|S|} \sum_{(u,i) \in S} |\hat{r_{ui}} - r_{ui}|$$

By comparing the RMSE and MAE values for our models, we were able to quantitatively evaluate the accuracy and effectiveness of our recommendation system models and compare their performance. In both metrics, a lower value indicates a better model fit.

4 Matrix Factorization-based Models

As we know, in matrix factorization methods, restaurant ratings are represented as a matrix $R \in \mathbb{R}^{m \times n}$, where m is the number of users and n is the number of restaurants in the dataset. We want to predict the unknown entries in this matrix by factorizing it into two lower-dimensional matrices, such that $R \approx PQ^T$: $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$, where k is the number of latent factors. The latent factors represent underlying characteristics that influence the ratings given by users to businesses.

Now, let's dive into the matrix factorization models we used in our recommendation system and how we evaluated them using the Surprise library in Python.

4.1 Baseline Method

As we learned in class, when predicting a missing rating for user u and item i, the matrix factorization model computes the dot product between the user's preference vector p_u and the item's characteristic vector q_i to estimate the rating, $\hat{r_{ui}} = q_i^T p_u$. However, in reality, some users may have a tendency to rate items higher or lower than others, regardless of their true preferences, and some items may receive systematically higher or lower ratings, independent of their actual quality. To account these systematic tendencies, we incorporated bias terms into the baseline method instead of using the dot product, as suggested in [7]:

$$\hat{r_{ui}} = \mu + b_u + b_i \tag{1}$$

This formulation predicts the rating $\hat{r_{ui}}$ for restaurant i by user u as the sum of the overall average rating μ, plus user and item bias terms b_u and b_i, respectively. The terms are calculated as follows:

$$\begin{cases} \mu = \frac{1}{|K|} \sum_{r_{ui} \in K} r_{ui} \\ b_u = \frac{\sum_{r_{ui} \in K_u} r_{ui}}{|K_u|} - \mu \\ b_i = \frac{\sum_{r_{ui} \in K_i} r_{ui}}{|K_i|} - \mu \end{cases} \tag{2}$$

where K is the set of all ratings in training set, K_u is the set of all ratings made by user u, and K_i is the set of all ratings received by business i.

4.2 FunkSVD

FunkSVD is an extension of the baseline method that also learns latent factors to capture the underlying structure of the rating data. To make predictions for missing ratings, the FunkSVD model combines the learned biases, the dot product between the user preference vector p_u and the item characteristic vector q_i, which captures the interaction between user u and item i, and the global average rating μ. The predicted rating for user u and item i is given by:

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u \tag{3}$$

To learn the model parameters (b_u, b_i, p_u and q_i), we initialize the user and item latent factor matrices according to a normal distribution with mean 0 and standard deviation 0.01. Then, we minimize the regularized objective function. The objective function consists of two terms: the first term is the sum of squared errors between the actual rating and the predicted rating, while the second term is the sum of squares of all the parameters, weighted by a regularization term λ

$$\sum(r_{ui} - \hat{r_{ui}})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{4}$$

To minimize this objective function, Simon Funk popularized a stochastic gradient descent (SGD) optimization during the Netflix Prize [3]. The algorithm iteratively updates the biases and latent factor vectors based on the error between the predicted rating and the actual rating ($e_{ui} \overset{def}{=} r_{ui} - \hat{r_{ui}}$). The parameters are updated by moving in the opposite direction of the gradient, yielding:

$$\begin{aligned} b_u &\leftarrow b_u + \eta(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \eta(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \eta(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \eta(e_{ui} \cdot p_u - \lambda q_i) \end{aligned} \tag{5}$$

where η is the learning rate. SGD is repeated over all the ratings until convergence.

Therefore, to achieve better prediction accuracy in the FunkSVD model, we have several parameters that can be tuned to achieve the best possible prediction accuracy for missing ratings.

First, the number of latent factors k (`n_factors`) can be adjusted based on the complexity of the rating dataset and the desired level of accuracy. A larger number of latent factors may lead to better prediction accuracy but also increase the risk of overfitting. On the other hand, a smaller number of latent factors may result in simpler models but may not capture all the underlying patterns in the data.

Second, the number of iterations (`n_epochs`) during training can affect the convergence of the model. A sufficient number of iterations is required to allow the model to learn the underlying patterns in the data but excessive iterations may lead to overfitting.

Third, the learning rate η (`lr_all`) in the SGD procedure can also impact the performance of the model. A large learning rate may cause the optimization process to miss the optimal parameters, while a small learning rate may result in slow convergence.

Finally, the regularization term λ (`reg_all`) can be adjusted to balance between the model's ability to fit the training data and its ability to generalize to new data. A high regularization term may result in a simpler model but may underfit the data, while a low regularization term may lead to overfitting.

4.3 SVD++

SVD++ is an extension of FunkSVD that incorporates implicit feedback to improve prediction accuracy. While independent implicit feedback is missing from our dataset, it is possible to capture this information by considering the set of restaurants that a user has rated, regardless of their specific rating. Essentially, a user's decision to voice their opinion and assign a rating (whether high or low) implicitly tells their preferences [6]. Hence, SVD++ adds a second set of restaurant factors, which relate each restaurant i to a factor vector $y_i \in \mathbb{R}^k$. These new restaurant factors are used to characterize users based on the set of restaurants that they rated. The predicted rating for user u and item i is then given by:

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T(p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j) \tag{6}$$

Here, $R(u)$ is the set of all restaurants rated by user u and a user u is modeled as $p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j$.

The regularized objective function and SGD procedure for SVD++ are similar to those for FunkSVD, with the addition of the new set of restaurant factors. The regularized objective function is:

$$\sum (r_{ui} - \hat{r_{ui}})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 + \sum_{j \in R(u)} y_j) \tag{7}$$

The user and item latent factor matrices p_u and q_i are also initialized according to a normal distribution with mean 0 and standard deviation 0.01, which is the same initialization approach used in FunkSVD. The SGD procedure updates the model parameters in the following way:

$$\begin{aligned} b_u &\leftarrow b_u + \eta(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \eta(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \eta(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \eta \left(e_{ui} \cdot (p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j) - \lambda q_i \right) \\ \forall j \in R(u) : y_j &\leftarrow y_j + \eta \left(e_{ui} \cdot |R(u)|^{-\frac{1}{2}} \cdot q_i - \lambda y_j \right) \end{aligned} \tag{8}$$

The parameters that can be tuned to improve prediction accuracy are the same as in FunkSVD: the number of latent factors, the regularization terms, and the learning rate and number of iterations of the SGD procedure. However, SVD++ also introduces a new set of item factors, which can be optimized to improve the model's ability to capture implicit feedback.

4.4 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) is a well-known algorithm similar to the basic matrix factorization technique. However, NMF introduces a crucial constraint that restricts the factor matrices to be non-negative. This restriction is critical since it allows the learned features to better represent user interests, community tendencies, and other actual meanings. The prediction in NMF is given by $\hat{r_{ui}} = q_i^T p_u$, which does not include the bias term. While the biased version of the model may yield better accuracy, prior research has demonstrated that it is highly prone to overfitting [9].

The optimization procedure in NMF uses a regularized SGD with a specific step size to ensure that factors remain non-negative, provided that their initial values are also non-negative. This constrained-optimization problem can be challenging to solve in practice; however, NMF addresses this issue by manipulating the learning-rate skillfully. The model is dependent on non-negative initial values, and the user and item factors are uniformly initialized in the range of [0, 1].

To enforce the non-negativity of the latent factors, NMF employs a hard regularization term in its optimization procedure. In contrast, SVD++ and FunkSVD use a soft regularization term to penalize deviations from the mean. The regularization term in NMF is given by:

$$\begin{aligned} p_{uf} &\leftarrow p_{uf} \cdot \frac{\sum_{i \in R(u)} q_{if} \cdot r_{ui}}{\sum_{i \in R(u)} q_{if} \cdot \hat{r_{ui}} + \lambda_u |R(u)| p_{uf}} \\ q_{if} &\leftarrow q_{if} \cdot \frac{\sum_{u \in U(i)} p_{uf} \cdot r_{ui}}{\sum_{u \in U(i)} p_{uf} \cdot \hat{r_{ui}} + \lambda_i |U(i)| q_{if}} \end{aligned} \tag{9}$$

Here, λ_u and λ_i are the regularization parameters for users and items, respectively. These parameters control the extent to which the model should regularize the factors. The regularization term ensures that the non-negative factors remain non-negative throughout the optimization procedure, thus enabling NMF to learn more interpretable and meaningful latent factors.

The parameters that can be tuned for NMF model are: the number of latent factors, the regularization terms of both users and restaurants, and the number of iterations of SGD procedure.

4.5 Evaluation

4.5.1 Default Parameters

Firstly, we evaluated the performance of FunkSVD, SVD++, and NMF using the built-in functions in the Surprise library with their default parameters.

- For SVD, the default latent factor was set to 100, the number of iterations and learning rate for SGD were set to 20 and 0.005 respectively, and the regularization parameter was set to 0.02.
- For SVD++, the default latent factor was set to 20, the number of iterations and learning rate for SGD were set to 20 and 0.007 respectively, and the regularization parameter was set to 0.02.
- For NMF, the default number of latent factors is set to 15, the number of iterations and learning rate for SGD were set to 50, and the default regularization parameters were set to 0.06.

The results are shown in Table 1.

Method	RMSE	MAE
Baseline	0.9929	0.7768
FunkSVD	1.0035	0.7821
SVD++	1.0025	0.7794
NMF	1.0710	0.8270

Table 2: Performance of Baseline Method, SVD, SVD++, and NMF with Default Parameters

We started with the default parameters since this allowed us to quickly assess the baseline performance of the models, and these default parameters are commonly used and have produced good performance in many experiments. However, as we can see from the evaluation results, the test RMSE and MAE of all three models with default parameters are worse than the baseline method, which only uses the bias term for rating prediction. Therefore, we need to tune the hyperparameters of these models to improve their prediction accuracy, and we will use GridSearchCV for this purpose.

4.5.2 Hyperparameters Tuning using GridSearchCV

GridSearchCV is an exhaustive search that trains and evaluates the model with all possible combinations of hyperparameters specified in the parameter grid. We evaluate the possible models with 5-fold cross-validation, and the parameter grids are as follows:

- For FunkSVD and SVD++:

Method	`n_factors`	`n_epochs`	`lr_all`	`reg_all`
FunkSVD	[20, 50, 80, 100]	[40, 60, 80, 100]	[0.002, 0.005, 0.007]	[0.02, 0.05, 0.1]
SVD++	[20, 50, 80]	[20, 40, 60]	[0.002, 0.005, 0.007]	[0.02, 0.05, 0.1]

- For NMF:

Method	`n_factors`	`n_epochs`	`reg_pu`	`reg_qi`
NMF	[15, 30, 50]	[30, 50, 80, 100]	[0.06, 0.1, 0.15]	[0.06, 0.1, 0.15]

We expected the larger range of latent factors for FunkSVD and SVD++ since these models have more complex structure compared to NMF and may benefit from more latent factors, as well as capture more complex relationships between users and items.

The results show that the performance of all three models was significantly improved after hyperparameter tuning compared to their performance with default parameters. The best hyperparameters found for each model and their corresponding RMSE and MAE are shown in the table below:

Method	Best Hyperparamters	RMSE	MAE
FunkSVD	n_factors: 20, n_epochs: 60, lr_all: 0.002, reg_all: 0.1	0.9900	0.7714
SVD++	n_factors: 20, n_epochs: 60, lr_all: 0.002, reg_all: 0.1	0.9901	0.7717
NMF	n_factors: 50, n_epochs: 30, reg_qi: 0.15, reg_pu: 0.15	1.0082	0.7717
Baseline		0.9929	0.7768

Table 3: Performance of Tuned Matrix Factorization Models and Baseline Method on Test Set

We can see that among the matrix factorization methods we evaluated, FunkSVD had the best performance, with the lowest RMSE and MAE. After tuning, both FunkSVD and SVD++ were able to achieve better results than the baseline method, while NMF did not and remained with the highest test RMSE. However, NMF's test MAE was still competitive with the other methods.

The best hyperparameters for FunkSVD and SVD++ were similar, with both models achieving their best performance with only 20 latent factors. However, NMF needed 50 latent factors to achieve its best performance, which was unexpected as we initially anticipated that the more complex models of FunkSVD and SVD++ would benefit from a larger range of latent factors.

The regularization terms also had a significant impact on the performance, with higher regularization leading to better results across all methods. These findings suggest that the trade-off between model complexity and regularization should be carefully considered in hyperparameter tuning.

Compared to the baseline method, the improvement in test performance is relatively small. One of the limitations of matrix factorization is the difficulty of incorporating side features, which limits its ability to make recommendations for users or items that are not present in the training set. Therefore, we decided to explore deep learning-based recommendation systems using FastAI as deep neural network models, which can address these limitations by easily incorporating query and item features to capture user interests and improve recommendation relevance.

5 Deep Learning-based Models

In this paper, we propose a deep learning-based collaborative filtering method that utilizes a feed-forward neural network architecture to predict star ratings for user-restaurant interactions.

Several studies have shown the effectiveness of deep learning-based collaborative filtering approaches for recommendation systems. Deep neural network-based approach has been implempented and outperformed traditional matrix factorization techniques on the MovieLens dataset [5]. By implementing this model, we expected that neural network architecture allows for creating highly customized models that can capture complex patterns in the data, and provide a more generalizable model to solve overfitting problem that is still present on the previous method.

Our approach involves minimizing the mean squared error (MSE) objective function using the Adam optimizer, and we incorporate an embedding layer and several continuous features to provide more information for the network to learn. Initially we built an embedding model based on collaborative filtering module by FastAI.

Then, feature engineering was conducted for building a more complex tabular model that can incorporate features other than only the user and restaurant IDs to provide more information for the network to learn. The proposed model was trained using best hyperparameters found by conducting tuning with Bayesian optimization. Models tested in this section are evaluated using the performance metrics RMSE and MAE.

5.1 EmbeddingDotBias Model

A PyTorch package called EmbeddingDotBias offers a fundamental dot model for collaborative filtering. This model implements embedding layers and the properties of a feed-forward neural network. Users and objects are represented as vectors in a low-dimensional space using embeddings. The `n_factors` parameter sets the number of latent factor for the embeddings size [12]. For each user-item pair, a prediction is computed using the dot product of these vectors and then added by user and item biases. Additionally, this model also create embeddings for user and item bias. The equations of this model are given below:

$$\hat{R} = PQ \tag{10}$$

$$\hat{r}_{u,i} = \sigma\left(q_i^T p_u + b_u + b_i\right) \cdot (r_{\max} - r_{\min}) + r_{\min} \tag{11}$$

where $\hat{r}_{u,i}$ is the star rating prediction, σ is the sigmoid function to transform the prediction output to the desired range and b_u, b_i are the bias terms for user and restaurant as mentioned in equation (2).

The MSE Loss function is given by:

$$\min_{P,Q,b} \sum_{(u,i)\in K} ||R_{u,i} - \hat{R}_{u,i}||^2 + \lambda\left(||P||_F^2 + ||Q||_F^2 + b_u^2 + b_i^2\right) \tag{12}$$

The model takes as inputs the number of unique users in the dataset `n_users`, the number of unique restaurants in the dataset `n_items`, and a range of possible output values `y_range`. In our case, the range of possible output values is set to 1 to 5 stars. To make a prediction for a user-item pair, the EmbeddingDotBias model first extracts the user and restaurant IDs from the input tensors. It then finds the embeddings for the users and items and calculates the dot product of the user and item embeddings. User and item biases are added to the dot product, allowing the model to capture user and restaurant item preferences that are independent of the embedding space. Finally, the output is transformed using the sigmoid function and mapped to the specified range. The model architecture is given as follow:

```
EmbeddingDotBias(
  (u_weight): Embedding(10728, 60)
  (i_weight): Embedding(5665, 60)
  (u_bias): Embedding(10728, 1)
  (i_bias): Embedding(5665, 1)
)
```

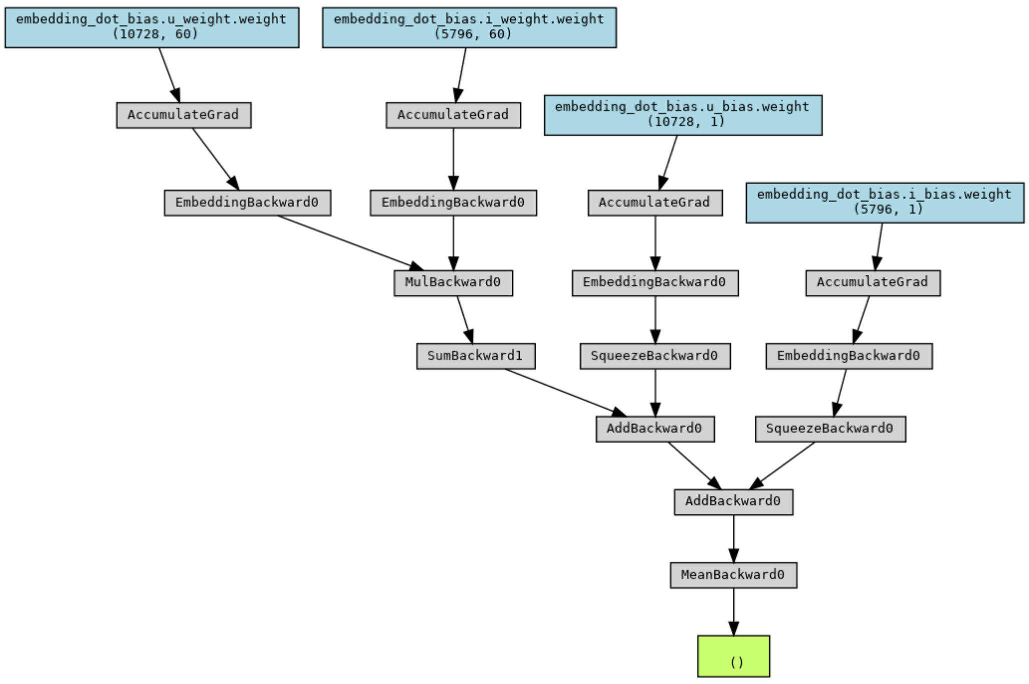

embedding_dot_bias.u_weight.weight (10728, 60)
embedding_dot_bias.i_weight.weight (5796, 60)
embedding_dot_bias.u_bias.weight (10728, 1)
embedding_dot_bias.i_bias.weight (5796, 1)
AccumulateGrad
EmbeddingBackward0
MulBackward0
SumBackward1
SqueezeBackward0
AddBackward0
MeanBackward0
()

Figure 2: EmbeddingDotBias Model visualization

Dataset were split with 80:20 ratio for training and testing data. Then the model was trained for 10 epochs using parameters as following: `n_factors = 60`, `valid_pct = 0.2`, `weight_decay = 0.1`, `y_range = (1,5)`, `learning_rate = 0.019`.

Epoch	Train loss	Val loss	Time
0	1.171339	1.171691	00:20
1	1.203682	1.219113	00:20
2	1.225976	1.227035	00:20
3	1.225370	1.216002	00:20
4	1.214949	1.193984	00:20
5	1.126273	1.155000	00:20
6	1.111838	1.120535	00:20
7	0.999374	1.081979	00:20
8	0.819021	1.069511	00:20
9	0.678870	1.069948	00:20

Table 4: EmbeddingDotBias Training result for 10 epochs

Test RMSE	1.032
Test MAE	0.832

Table 5: EmbeddingDotBias Test RMSE and MAE

The test results for the EmbeddingDotBias model showed a test RMSE of 1.032 and a test MAE of 0.832. However, it is worth noting that the model appeared to be overfitting the data. Despite a significant decrease in the training loss, the model's test RMSE remained high, indicating that it may not generalize well to new data.

5.2 Tabular Neural Network Model

The tabular neural network is a more complex feedforward neural network that allows us to incorporate additional features beyond user and restaurant IDs. It is implemented in the FastAI library and is designed to work with tabular data. In our case, the tabular data consists of the user ID, restaurant ID, and additional features. This model addressed the limitation of all the previous matrix factorization based model which the inputs are only limited to user and restaurant IDs.

To address the issue of overfitting in the EmbeddingDotBias model, we decided to incorporate additional features into our deep learning model. Specifically, we retrieved the user and restaurant biases from the EmbeddingDotBias model and appended them as additional features to our tabular neural network model.

The user and restaurant biases were obtained from the u_{bias} and i_{bias} layers of the EmbeddingDotBias model, respectively. We used these biases as additional features in the tabular neural network model to capture more information about user and restaurant preferences and improve the model's performance. Other additional features related to the restaurants are also incorporated as the input for this neural network.

The input data for our recommender system consists of both categorical and continuous variables. The categorical variables include user IDs and restaurant IDs, while the continuous variables include the average star rating and review count for each restaurant, as well as the average sentiment score of the restaurant reviews. In addition, we also incorporated user and restaurant biases as additional features derived from the EmbeddingDotBias model.

To prepare the data for training our deep learning models, we performed several preprocessing steps. For the categorical variables, we used the categorify function from the FastAI library to convert the categorical variables into integer-encoded values. For the continuous variables, normalized the variables using the mean and standard deviation of the training set. It is important to note that we only normalized the continuous variables for the training set to prevent data leakage. The model architecture is given as follow:

```
TabularModel(
    (embeds): ModuleList(
        (0): Embedding(10728, 289)
        (1): Embedding(5679, 203)
    )
    (emb_drop): Dropout(p='emb_p', inplace=False)
    (bn_cont): BatchNorm1d(5, eps=1e-05, momentum=0.1, affine=True)
    (layers): Sequential(
        (0): LinBnDrop(
        (0): Linear(in_features=497, out_features='layer1_output', bias=False)
        (1): ReLU(inplace=True)
        (2): BatchNorm1d('layer1_output', eps=1e-05, momentum=0.1, affine=True)
        (3): Dropout(p='layer1_dropout', inplace=False)
        )
        (1): LinBnDrop(
            (0): Linear(in_features='layer1_output', out_features='layer2_output', bias=False)
            (1): ReLU(inplace=True)
            (2): BatchNorm1d('layer2_output', eps=1e-05, momentum=0.1, affine=True)
            (3): Dropout(p='layer2_dropout', inplace=False)
        )
        (2): LinBnDrop(
            (0): Linear(in_features='layer2_output', out_features=1, bias=True)
        )
        (3): fastai.layers.SigmoidRange(low=1, high=5)
    )
)
```

The model architecture consists of three main components: embedding layers, fully connected layers, and a sigmoid activation function. The embedding layers are used to convert the categorical user and restaurant IDs into dense vectors that can be processed by the fully connected layers. The `emb_drop` layer is a dropout layer that randomly sets a fraction of the input elements to zero to the output of the embeddings during training to prevent overfitting.

The continuous features are passed through a batch normalization layer `bn_cont` to normalize the input values.The fully connected layers of the model are defined in the layers sequential block, with three linear layers, the first 2 layers are followed by a ReLU activation function, batch normalization, and dropout. The first linear layer takes as input the concatenated embeddings and continuous features, and has an output size of `layer1_output`. The second linear layer has an output size of `layer2_output`, while the final linear layer has an output size of 1. For the first two linear layers, it is also followed by a batch normalization layer with shape of layer1_output and layer2_output', respectively. It applies the same for the dropout layers with the value layer1_dropout and layer2_dropout. The SigmoidRange activation function maps the output of the model to a range of 1 to 5, which corresponds to the possible star ratings in our dataset.

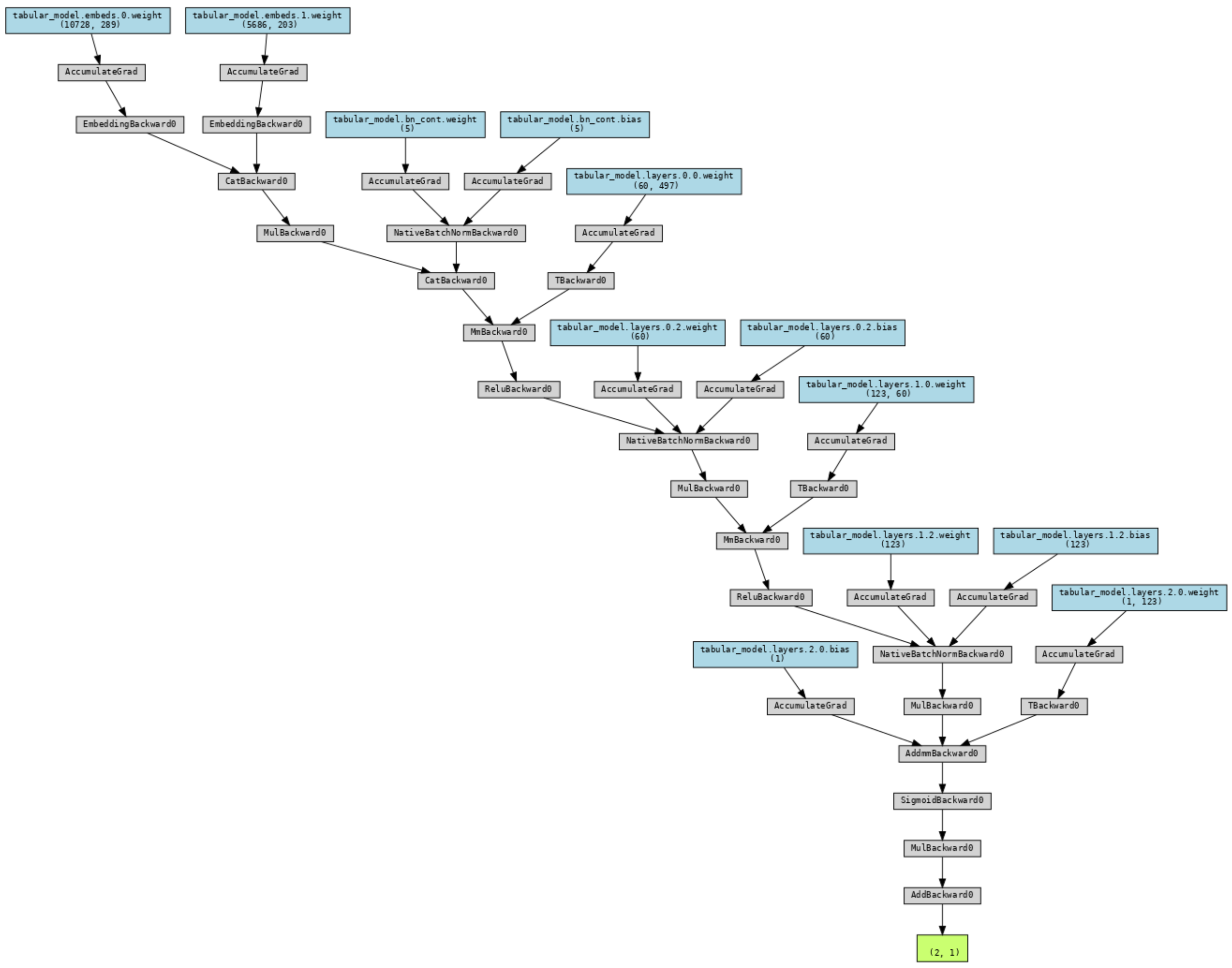


Figure 3: Tabular Model visualization

5.2.1 Base Model

Hyperparameters: batch size = 128, valid pct = 0.2, embed_p = 0.2, layer1_output = 60, layer2_output = 100, layer1_dropout = 0.3, layer2_dropout = 0.1, weight_decay = 0.1, learning_rate = 0.0017

Epoch	Train loss	Val loss	Time
0	0.685757	1.104964	00:17
1	0.781145	1.024006	00:17
2	0.829000	0.997748	00:17
3	0.817950	1.001257	00:17
4	0.779931	1.020261	00:17
5	0.746451	1.060026	00:17
6	0.684708	1.115694	00:17
7	0.612241	1.142692	00:17
8	0.555963	1.166991	00:17
9	0.519988	1.190424	00:17

Table 6: Tabular Base Model Training result for 10 epochs

Test RMSE	1.087
Test MAE	0.817

Table 7: Tabular Base Model Test RMSE and MAE

5.2.2 Hyperparameters Tuning using Bayesian Optimization

Considering that the overfitting still persist in our tabular neural network base model, we conducted hyperparameter tuning to find the optimal values for a set of hyperparameters that control the behavior of the model during training. We implemented Bayesian optimization, which is a probabilistic approach to hyperparameter tuning that uses a combination of prior knowledge and observed data to guide the search for the optimal hyperparameters. Bayesian optimization in turn takes a prior distribution of the function to be optimized and updates it based on the most promising validation scores [1].

In neural network models, which the training process are computationally expensive, Bayesian Optimization is very efficient compared to Random Search or Grid Search since it takes into account the previous evaluation with the goal of minimizing the objective function MSE using a set of hyperparameters defined in a searching space.

We used Bayesian optimization to find the optimal hyperparameters for our tabular neural network model with stratified 5-fold cross-validation. This involved splitting our data into 5 training and validation sets while preserving equal target value distribution across all sets and using each set in turn as a validation set while training the model on the remaining data.

The hyperparameters we optimized included the learning rate, weight decay, layer1 dropout rate, layer2 dropout rate, and output shape of the second layer. By using Bayesian optimization with 5-fold cross-validation, we were able to efficiently explore the hyperparameter space and find the optimal values for our model. Result of best hyperparameters: `layer1_dropout: 0.5`, `layer2_dropout: 0.3`, `layer2_output: 123`, `learning_rate: 0.0077565738454144564`, `weight_decay: 0.035247792719299315`

5.2.3 Tuned Model

Using the best hyperparameters found, we trained a new model for 10 epochs. We also introduced a callback feature to reduce learning rate if the validation loss does not improve, to prevent the model to diverge.

Epoch	Train loss	Val loss	Time
0	0.934541	0.943029	00:17
1	0.934699	0.941350	00:19
2	0.941237	0.936138	00:18
3	0.924801	0.923352	00:18
4	0.928668	0.935248	00:18
5	0.866740	0.946272	00:18
6	0.880639	0.954821	00:18
7	0.833089	0.987147	00:18
8	0.796797	1.016733	00:17
9	0.739466	1.027347	00:17

Table 8: Tabular Tuned Model Training result for 10 epochs

Test RMSE	0.959
Test MAE	0.745

Table 9: Tabular Tuned Model Test RMSE and MAE

5.2.4 Features Importance

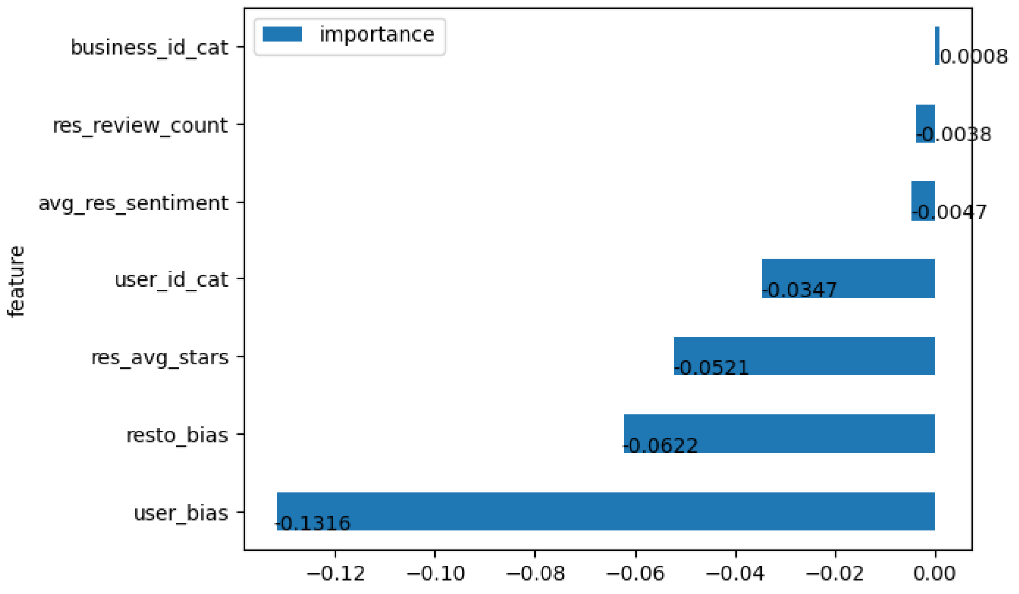


Figure 4: Features Importance Plot

We used Permutation Importance to analyze the feature importance within the tabular model. The idea of permutation importance is to measure the predictive performance of the model when a particular column, thus feature, within the dataset is randomly shuffled. We do the shuffling individually for all attributes in the dataset. The importance value is measured by the equation

$$\text{importance} = \frac{\text{base error} - \text{current error}}{\text{base error}}$$

Base error refers to the error of the model previous to the shuffling. Therefore, a lower value of importance shows that the feature has more predictive power to the target attribute.

The final results show user's bias having the lowest value of importance. Thus, within this model, user's bias has a large impact on the model's ability to predict accurately the target attribute. However, it is important to note that permutation feature importance does not reflect the intrinsic predictive ability of the feature itself, but it is specific for each feature in this specific model.

6 Result and Discussion

Metrics	FunkSVD	SVD++	NMF	EmbedDB	TabularBase	TabularTuned
Test RMSE	0.990	0.990	1.008	1.032	1.087	0.959
Test MAE	0.771	0.772	0.772	0.832	0.817	0.745

Table 10: Models Performance Result based on Evaluation Metrics

The table above compares the performance of our collaborative filtering models with tuned hyperparameters towards the test set. The deep learning model achieved the lowest RMSE and MAE values by a significant margin, outperforming the FunkSVD, SVD++, and NMF models. This shows that the deep learning model has the potential to generate more accurate recommendations, which would improve user experience.

Our deep learning-based collaborative filtering method, which utilizes a tabular neural network architecture, offers several advantages over conventional matrix factorization techniques performed in the Matrix Factorization-based techniques. One of the key benefits is the ability to incorporate more features into the model, which can reduce bias predictions and improve the accuracy of star rating predictions. By providing information about restaurant review count, average star ratings from users, and average sentiment scores of restaurant reviews, we were able to provide more information for the neural network to learn and make more accurate predictions.

In addition, our tabular neural network architecture is more complex, allowing us to capture more information about the data and discover various relationships between different inputs. By using activation functions on hidden layers, we were able to capture non-linear and complex relationships from the input data, converting the input into a more understandable prediction. Our use of dropout layers in the network structure also helped to create a more generalizable model, preventing overfitting compared to the embedding model.

7 Conclusion

The results of this study demonstrate that Matrix Factorization-based and Deep Learning-based collaborative filtering methods are effective in predicting user ratings for restaurant recommendation systems. Furthermore, the deep learning model outperformed other models with the lowest MAE and RMSE. The deep learning model utilized features such as average restaurant stars, restaurant review count, average restaurant sentiment from sentiment analysis on past text reviews using VADER, user bias, and restaurant bias, providing more information to the network. In addition, complex structure, together with regularization and dropouts, help the model to improve its prediction performance on unseen data.

8 Future Work

The future work of the restaurant recommendation system could focus on improving the model by incorporating additional restaurant and user features. There are more restaurant features, such as restaurant categories, that could be included in the model. Furthermore, implementing Natural Language Processing technique to preprocess restaurant categories and attributes, then transforming them into vectorized texts, could also improve the prediction quality. This approach could provide the model with information about similar restaurants that a user likes and discover new relationships from the additional information. Further analysis of users' social connections, such as friends and the number of fans, could also potentially improve the model by exploring tie strength between users.

References

[1] *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, 2010.

[2] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40:66–72, 03 1997.

[3] Simon Funk. Netflix update: Try this at home, 2006.

[4] C. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the International AAAI Conference on Web and Social Media*, 8:216–225, 05 2014.

[5] International Conference on Machine Learning. *A Neural Autoregressive Approach to Collaborative Filtering*, 2016.

[6] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model, 2008.

[7] Yehuda Koren. Factor in the neighbors. *ACM Transactions on Knowledge Discovery from Data*, 4:1–24, 01 2010.

[8] Pasquale Lops, Marco Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*. Recommender Systems Handbook. Springer, 2011.

[9] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10:1273–1284, 05 2014.

[10] Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence. *Empirical analysis of predictive algorithms for collaborative filtering*, 1998.

[11] J. Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. *Proceedings of the 1st ACM conference on Electronic commerce - EC '99*, 1999.

[12] Abebe Tegene, Qiao Liu, Yanglei Gan, Tingting Dai, Habte Leka, and Melak Ayenew. Deep learning and embedding based latent factor model for collaborative recommender systems. *Applied Sciences*, 13:726, 01 2023.

Appendices

A Source Code, Docs & Datasets

All source code files, documents, and datasets used in the project are publicly accessible through the following OneDrive link: https://portland-my.sharepoint.com/:f:/g/personal/dkyennoto2-c_my_cityu_edu_hk/EsoI4CVaCy9Hsn-60brXEiMBZiYlns7LJnYrkAVuqEg7JQ?e=ZYrYSA

For more detailed information on how to run the code, please refer to the `README.md` file located in our GitHub: https://github.com/ronaldgustaf/SDSC3002-Community-README. The README file provides step-by-step instructions on how to run the code, including how to set up the environment and execute the notebooks.