DSA4212 Assignment 2
Group 13

Nicholas Russell Saerang, A0219718W, e0550376@u.nus.edu
Clifton Felix, A0219735X, e0550393@u.nus.edu
Jason Ciu Putra Sung, A0219734Y, e0550392@u.nus.edu

1 Introduction

This project aims to predict the rating of films given by users. The data used in this project consists of 3 CSV files: train, test, and anime data. Anime data contains the details of each anime, while the train and test contain the rating of films given by users. We'll be using train and anime data during the training process and measuring the model performance (MSE) on test data. In this report, the terms movie, anime, film, and item are used interchangeably.

2 Experiments

In the first stage, we will try various non-factor models such as naive models, content-based models, and regression models. In the second stage, we delve into collaborative filtering, which uses some similarity metrics and nearest-neighbour techniques, but it all boils down to factor models or matrix factorization-based models.

2.1 Naive Model

2.1.1 Random Model

2.1.1.1 Uniform Random Integer

The first model we experimented was Random Model that assigns a uniform random integer from 1 to 10 to each row in test data. With this model, we achieved **17.02** test MSE.

2.1.1.2 Normal Random Integer

Another random model we experimented with was Random Model that assigns a normal random integer with mean 7.81 and standard deviation 1.57. The mean and standard deviation were obtained by fitting a normal distribution on the ratings in train data. With this model. With this model, we achieved a much lower **4.95** test MSE.

2.1.2 Item Mean Model

Since the anime data has a rating column, we tried building an item mean model using item means in both anime data and train data.

2.1.2.1 Using Item Mean in Anime Data

Since not all anime IDs in train data exist in Anime data, we imputed the NAs with the mean rating of all animes (6.474). With this model, we achieved **2.08** test MSE.

2.1.2.2 Using Item Mean in Train Data

Another variant we tried is using the item mean in train data. Since there are some anime IDs in test data that are not inside train data, we imputed the NAs with the mean rating of the aggregated item mean in train data

(6.678). With this model, we achieved **2.06** test MSE.

2.1.3 User Mean Model

We also tried building a user mean model where we predict the ratings given by a user based on the mean of the ratings they gave in train data. We then imputed all NAs with the mean rating of the aggregated user mean in train data (8.216). With this model, we achieved **1.86** test MSE.

2.1.4 Regularized User Mean Model

We also tried using a regularized estimate $\hat{R}_{u,f} = \frac{1}{\alpha+|F(u)|}(\alpha M + \sum_{f \in F(u)} R_{u,f})$, where M is the overall training mean (7.81), $F(u)$ is the set of all animes watched by user u, and α is a hyperparameter. Whether to impute all NAs with the aggregated user mean (8.216) or the overall training mean (7.81), over a list of values of α, the best test MSE we can obtain is **2.06**. Surprisingly, using regularization doesn't make our model performs better.

2.2 Content-Based Model

The next model we tried is Content-Based model. Before building the model, we preprocessed the anime data:

1. Combined name and genre
2. Applied text cleaning (removing numbers, punctuations, and lowercasing text)
3. Generated the TF-IDF vectors
4. Imputed episodes "Unknown" with the mean episodes
5. Scaled episodes, ratings, and members to range (0, 1)
6. Dummified the type

After all these preprocessing steps, our anime data is of size (12294, 11862).

For content-based models, there are some ways to find similar items, such as using cosine similarity, k-nearest neighbours, etc. However, due to memory limit and time constraints, cosine similarity and k-nearest neighbours were not possible. Hence, after some research, we came across the annoy library which implements an approximation of KNN (Approximate Nearest Neighbours, ANN). This library is used by Spotify for music recommendations.

According to [Spo], ANN is implemented by constructing trees and employing random projections. A random hyperplane is selected at each intermediate node in the tree, dividing the space into two subspaces. By selecting the hyperplane that is equally spaced from two points from the subset, this hyperplane is determined. This process is repeated until we get a forest of k trees.

Using this library, we obtained 100 approximate nearest neighbours for each item based on the cosine similarity score. The predicted rating of film i given by user u (r_{ui}) is:

$$\hat{r}_{ui} = \frac{\sum_{j \in Ru \cap j \in ann_{100}(i)} r_{uj}}{|\{j \in R_u | j \in ann_{100}(i)\}|} \quad (1)$$

where R_u is the items rated by user u and $ann_{100}(i)$ is the 100 approximate nearest neighbours of item i.

For user u that has never rated any of the 100 neighbours of item i, we imputed the rating with the user mean rating in train data if the user has rated at least once, else we imputed it with the mean rating of the aggregated user means in train data (8.216).

With this model, we achieved test MSE of **1.85** which is slightly better than the user mean model.

After some tuning, an improvement was made when we modified the formula of $\hat{r}_{ui}$ to be the weighted average instead of the simple average with **fifth power** of the cosine as the weights.

$$\hat{r}_{ui} = \frac{\sum_{j \in Ru \cap j \in ann_{100}(i)} s^5(i,j) r_{uj}}{\sum_{j \in Ru \cap j \in ann_{100}(i)} s^5(i,j)} \quad (2)$$

where $s(i,j)$ is the cosine similarity of item i and j.

With this model, we achieved **1.78** test MSE.

Then, another improvement was made when we considered 500 approximate nearest neighbours and used the **ninth power** of the cosine as the weights.

$$\hat{r}_{ui} = \frac{\sum_{j \in R_u \cap j \in ann_{500}(i)} s^9(i,j) r_{uj}}{\sum_{j \in R_u \cap j \in ann_{500}(i)} s^9(i,j)} \tag{3}$$

With this model, we achieved **1.67** test MSE.

2.3 Regression

We then tried to engineer some features for each user ID and anime ID and use regressor models to predict the ratings with the engineered features as predictors. In this section, we tried 3 regressor models: Linear Regression, XGBoost Regressor, and neural network.

2.3.1 First Variant

The first variant we tried was using 2 predictors, user mean rating and movie mean rating, to predict ratings. The movie mean used here is the rating column of Anime data and not the aggregated movie mean rating of train data. For anime IDs in train and test that are not inside Anime data, we imputed the movie mean with the mean rating of all animes (6.474), while for user IDs in test data that are not inside train data, we imputed the user mean with the mean rating of the aggregated user means in train data (8.216).

The fitted Linear Regression model is

$$\begin{aligned}\hat{\text{rating}} = &-6.265 + 0.874 \times \text{movie\_mean} \\ &+ 0.943 \times \text{user\_mean}\end{aligned}$$

and this model achieved **1.51** test MSE. With XGBoost Regressor, we achieved **1.49** test MSE.

2.3.2 Second Variant

We tried generating some other features and testing their performance. The best set of features we found are: user mean, movie mean, number of movies user rated, number of users who rated the movie, user and movie minimum and maximum rating, and anime data (all columns combined except name).

We first did the necessary preprocessing for genre and type, and imputation for episodes. We then built the dataframe with the specified features for training. With Linear Regression, we achieved **1.5** test MSE, while with XGBoost Regressor, we achieved **1.46** test MSE.

We also fitted a neural network with this set of features, adapted from [Nad20]. For the neural network, we performed some additional preprocessing, such as scaling members, movie mean rating, and episodes to range (0, 1), and label-encoding anime_id and user_id. The label encodings are needed as the IDs are not consecutive.

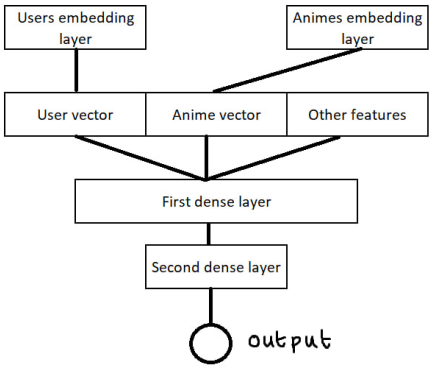

Figure 1: Neural Network

Subsequently, we built a neural network that consists of an input layer of size 260, 2 hidden layers of 128 and 32 neurons with ReLU activation, and a final layer to produce the output. The input layer is a combination of anime ID and user ID embeddings, each of size 100, and 60 other neurons for 60 other features. The optimizer used is Adagrad with learning rate of 0.003, and the batch size is 4096. After training for 100 epochs, we achieved **1.43** test MSE.

2.4 Collaborative Filtering

According to [Dey22], collaborative filtering is a recommendation systems method that is solely based on the past interactions that have been recorded between users and items, in order to produce new recommendations. For the context of this project, we build systems solely based on the available ratings made by users on animes and try to find which animes other users similar to the user of interest would prefer. The definition makes it different from our previously experimented content-based method which uses additional information such as the anime descriptions or the anime genres.

2.4.1 Cosine Similarity

We generated a sparse version of the full train rating matrix, and then we computed the pairwise cosine similarity between two users as demonstrated in [Ajm17]. These similarities make up the weights for the prediction later on, as the prediction is based on the combination of ratings of people who have watched the same anime of interest. If the user ID is not found in the matrix, we will use the average rating of the anime taken from the anime data. A fallback option is if the anime is also not found in the matrix, to which we will predict the overall training data mean instead. After running this prediction on the test set, we obtained a test MSE of **1.86**.

2.4.2 Simple Matrix Factorization

Next, we tried the matrix factorization model, which according to [Che20], involves creating a matrix to represent user-anime interactions and then factorizing this matrix into two vectors, one for users and the other for films. These vectors, called latent factors, capture the latent features or attributes that can predict how much a user will like a specific film. Once we learn these factors, we can simply compute the outer product to create a matrix of all the user-film rating combinations.

The factor model we used approximates two latent factors that were initialized as normally distributed random vectors. It employs Adam Optimizer and Gradient Descent to get the outer product of these vectors as close as possible to the full matrix, in terms of mean squared error. We then created a larger model consisting of two factor models, both correlated to each other. However, we did not create the full matrix in the process, as it is too large. Instead, we only kept the necessary matrix indices and entries in the form of lists.

2.4.2.1 Base model

The first one is the base factor model. It tries to model the full matrix as an outer product of the 2 latent factors. Here we tried to use both Gradient Descent and Adam Optimizer to get the best possible factors and compared the results. Both models performed very well, achieving a test MSE of **1.49** each.

2.4.2.2 Residual model

To further improve on the base models, we try to model their residuals as well, so that we can minimize it using another pair of latent vectors. It has the same methodology as the base model, with the only difference lying in the matrix that we are trying to model. Having retrieved the new residual factors using different combinations of optimizers, and then adding them to the base model, we were able to achieve a test MSE of **1.45** (combining Adam base model and GD residual model).

2.4.2.3 Halving technique

To our surprise, we discovered that we can create the complete matrix and use it during training by reducing its precision by half. By default, NumPy computes the entries of the matrix as 64-bit numbers. Therefore, converting them to 32-bit numbers allows us to have

the same matrix occupying much less memory. This is possible because the entries of the matrix are not so large that they require a 64-bit representation.

Using JAX, we trained the latent vectors using three methods: **sequential gradient descent** (4000 iterations, learning rate of 40, update weight vector then anime vector) with test MSE of **1.59**, **parallel gradient descent** (4000 iterations, learning rate of 35, update both latent vectors at the same time) with test MSE of **1.60**, and **L-BFGS** (maximum iteration of 1000, with the help of SciPy) with test MSE of **1.48**.

2.4.3 Singular Value Decomposition

Singular Value Decomposition (SVD), according to [Bag22], decomposes a user-film interaction matrix into three lower-dimensional matrices: a user matrix, a diagonal matrix containing singular values, and a film matrix. Like matrix factorization, these matrices capture the latent factors or attributes that influence a user's preference for a film. SVD has proven to be very effective in dealing with sparse datasets, but the main challenge arises when the dataset is too large, as it is in our case. Fortunately, there is a Python library called `surprise`, which can be seen on [Hug] and, created specifically for recommendation systems, which we used to implement our SVD model. The full documentation of the SVD model is available at [svd].

We first used GridSearchCV which performs a grid search to find the best hyperparameters for SVD. By running it on 3 folds of cross-validation, we managed to get the best hyperparameters for the model, which are learning rate of 0.01, regularization term of 0.06, and number of factors of 500. With these hyperparameters, SVD model achieved **1.24** test MSE, which is the best result that we can get so far.

3 Bibliography

[Ajm17] Ajmichelutti. Collaborative filtering on anime data, Mar 2017. URL: https://www.kaggle.com/code/ajmichelutti/collaborative-filtering-on-anime-data.

[Bag22] Reza Bagheri. Understanding singular value decomposition and its application in data science, Oct 2022. URL: https://towardsdatascience.com/388a54be95d.

[Che20] Denise Chen. Recommendation system-matrix factorization, Jul 2020. URL: https://towardsdatascience.com/d61978660b4b.

[Dey22] Victor Dey. Collaborative filtering vs content-based filtering for recommender systems, May 2022. URL: https://analyticsindiamag.com/collaborative-filtering-vs-content-based-filtering-for-recommender-systems/.

[Hug] Nicolas Hug. Surprise. URL: https://surpriselib.com/.

[Nad20] Nadergo. Hybrid anime recommendations with pytorch, Jun 2020. URL: https://www.kaggle.com/code/nadergo/hybrid-anime-recommendations-with-pytorch.

[Spo] Spotify. Spotify/annoy: Approximate nearest neighbors in c++/python. URL: https://github.com/spotify/annoy.

[svd] Matrix factorization-based algorithms. URL: https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD.