ABSTRACT

Recommender System is an essential topic in machine learning as it influences many industries in times like this. In this study, we used the dataset of Netflix Prize and compared the performance of four models: Matrix Factorization with ALS (both with or without bias), AutoRec System (both item-based and userbased), Naive Bayes Recommender System, and Neural Collaborative Filtering.

KEY WORDS: Matrix Factorization; AutoRec; Naive Bayes; Collaborative Filtering

CONTENTS

1 Introduction

A recommendation system is an algorithm designed to recommend related items to users. Recommendation systems are essential in specific industries because they can effectively generate significant revenue or can be a way to stand out from competitors. Netflix organized a challenge (the ”Netflix prize”) where the goal was to produce a recommender system that performs better than its algorithm, which proves how important recommender systems have become.

The first section of the report will briefly introduce the background of the recommender system, the objectives of this project, and the problem formation process. The second section presents the exploratory data analysis to visualize different features in the provided dataset and gives detailed explanations on data preparation and parameter selection. The third section provides illustrations on all of the models I constructed for the dataset, focusing on a more technical perspective. The fourth section discusses the model results and problems encountered with possible solutions towards them. The fifth section concludes the comparative results of models and feasible future improvements on the system.

1.1 Background

The growing role of the Internet as a forum for electronic and business transactions has accelerated the advancement of recommender device technologies. The simplicity with which the Web allows users to have suggestions on their preferences and dislikes is a significant catalyst in this regard. Consider a situation in which a service company, such as Netflix, is involved. Users can quickly provide suggestions in these situations by simply clicking a button. Users choose numerical values from a particular ranking system (e.g., a five-star rating system) to determine their likes and dislikes of different products, a standard methodology for providing reviews [1].

The fundamental principle behind recommender systems is to infer consumer needs from a variety of data sources. The entity to which the recommendation is provided is referred to as the user, and the product being recommended is often referred to as an item. As a result, since previous preferences and proclivities are also effective measures of potential decisions, recommendation analysis is frequently focused on prior interactions with users and items [1].

The fundamental premise of recommendations is that there are significant dependencies between user-centric and item-centric operations. Various groups of items can display substantial similarities in certain situations, which can be used to make more precise recommendations. Alternatively, rather than divisions, dependencies can exist at the finer granularity of individual items. The rating matrix can be used to learn these dependencies in a data-driven manner, and the resulting model is used to create predictions for target users. The more significant the amount of rated objects available to a user, the simpler it is to make reliable assumptions about the user's potential behavior [1].

1.2 Objectives

This paper would use the Netflix dataset to build recommendation systems that forecast user ratings for movies they have not seen yet. The highest expected ratings are suggested to the user. My objective is to compare the performances of different models by comparing the RMSE scores. Though it is not the only factor influencing the recommender system's efficiency, increasing the precision of these expected scores would undoubtedly demonstrate that consumers' preferences can be better understood. As a result, the RMSE consistency score would be one of the most critical metrics for testing my models.

1.3 Problem Formation

Given the Netflix Prize dataset, I have come up with different model choices, and my focus will be on comparing the performances of these models with such an extensive dataset.

In this project, I implemented models of matrix factorization with alternating least square algorithm (MFALS), matrix factorization with Auto Encoder (AutoRec), Naïve Bayes Collaborative Filtering (NBCF), and neutral Collaborative Filtering (NCF) on Netflix Dataset. For the first model, I abbreviate Matrix Factorization with Alternating Least Squares as MFALS and Biased Matrix Factorization with Alternating Least Squares as BMFALS.

MFALS is a classic recommender framework model that ensures high accuracy of forecast ratings and rapid convergence speed. When studying, it continues updating the matrix and can fill in missing values by multiplying two latent matrices, U and V.

Another way to apply matrix factorization to movie data is to use AutoRec (also known as Auto Encoder). AutoRec, unlike ALS, does not need to produce the whole rating matrix. Instead, it compresses the sparse vectors of users and items before decoding them back to their original shape. Since the batches mechanism feeds data part by part, it utilizes fewer resources for testing per epoch as a recommender system, so the memory overload issue can be eliminated by training the model on a portion of the dataset at a time. In addition, because of the model's simplicity, it uses fewer computing resources during testing, resulting in a reduction in training time.

The Naive Bayes Collaborative Filtering approach is based on the Bayes theorem. It is understood that Bayesian classification methods are constructed using the user rating for the item and item knowledge to create a model. It is believed that attributes are conditionally autonomous in the Naive Bayes classifier. Each attribute value's probability of being assigned into a class is calculated, as well as the prior probability of each class. The class label of a new tuple or instance may be calculated using the expected probabilities. Moreover, if the device has more sparsity, it can have greater scalability and consistency recommendation results. Furthermore, it saves time by building a model offline and then calculating the suggestion results online. Bayesian classifiers outperform decision trees in terms of accuracy and speed, mainly when applied to large datasets.

The Neural Collaborative Filtering is designed for solving the problem of making recommendations based on having potential feedback. Although some work has been done to apply deep learning into recommender systems, deep learning is mainly used to build models on auxiliary information, for example, the projection of detecting words and images, the voice feature of music. When considering the critical factor, the cross between the user and item, deep learning still uses matrix factorization, using the inner product of the user and item to be the featured product. Using the neural network structure to replace the inner product can learn any function from the dataset, this is what I called Neural Collaborative Filtering, NCF is a structure that can express and advance the matrix factorization, and I have pointed out the MLP model to learn the interaction function between user and item.

2 Data

The data I use to train and test the model is the dataset from the Netflix Prize Dataset.

2.1 Description

The Netflix Prize Dataset has a training set, a qualifying set, and a test set. In the Netflix Prize competition, the contestants are asked to train their model on the training set, get the RMSE score from the probe set, and submit a prediction list. The users' ratings in the test set are never revealed, so the contestants cannot adjust their model according to the test set for better performance, which improves the quality of their models. However, in my implementation, I will use the data from qualifying.txt as my test data. The training set and the probe set are contained in four files called combined_data_1.txt,combined_data_2.txt, etc. .The original dataset also has a file containing all the names of the movies, which is not used in my implementation. I separate the training set and test set first for exploratory data analysis and further modeling.

There are in total 100,480,507 ratings from 480,189 different users in the dataset who rated 17,770 different movies. Next, I will explore the datasets from different aspects.

2.1.1 Movie Release Date

The number of movies released is stable at around 0 during the period from 1900 to around 1930. Then the number slowly starts to go up after 1930. After 1980, the number of movies released each year started to rocket. Most movies are released around the time of 2000 in this dataset. This trend is in line with the development of the movie industry as well. The distribution of movie release dates is shown in Figure 2-1.

This millennial has witnessed a large number of Netflix movies released. However, the plot does not reveal whether Netflix favors young films or no older films. The sharp decline of the rightmost point is most likely due to an incomplete last year.

2.1.2 Rating

Users rarely rate a movie to be 1 or 2. Most ratings are above 3 and 4 is the rating that is most frequently given. Since only those who like the movies become consumers, the distribution is likely biased, and others would presumably abandon the channel. The distribution of the rating is shown in Figure 2-2.

This may be attributed to the fact that disappointed consumers are more likely to abandon

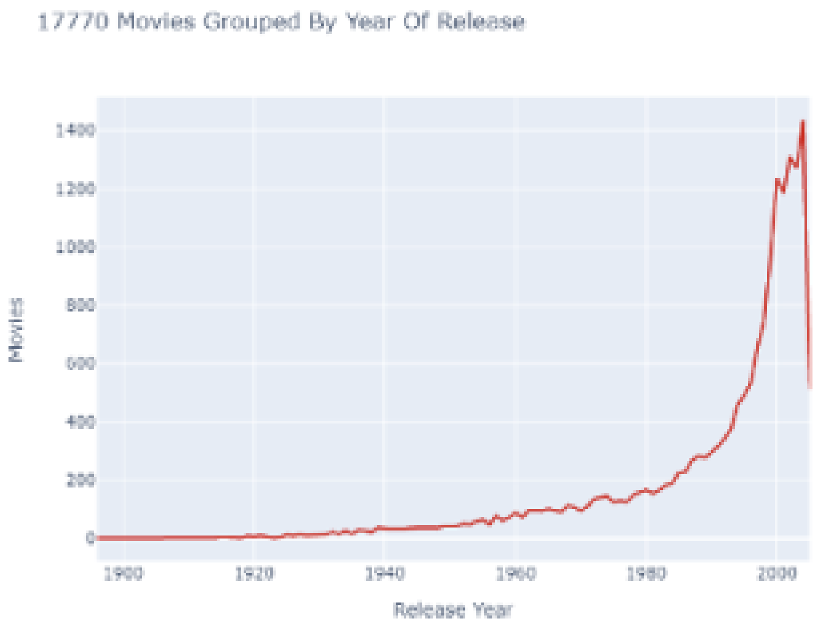

Figure 2-1 Distribution of Movie Release Date

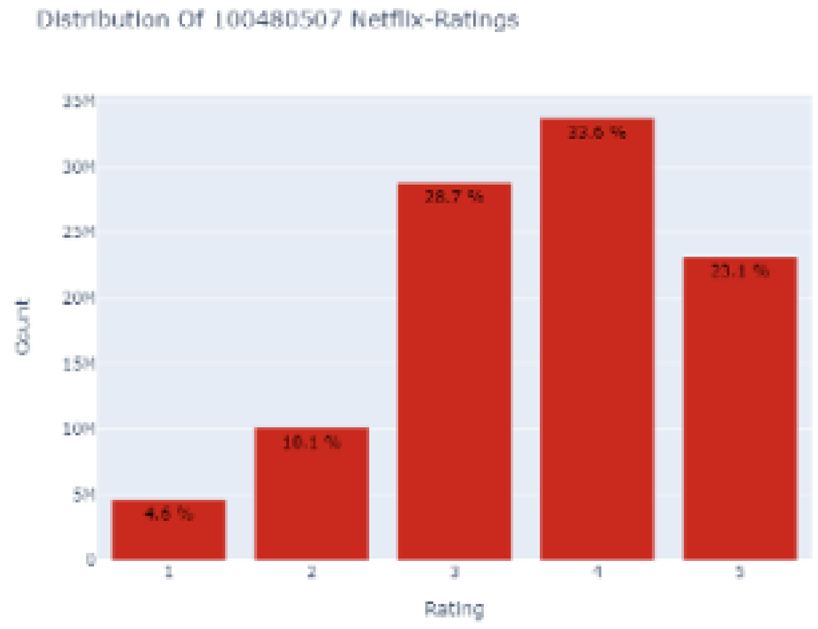

Figure 2-2 Distribution of Rating

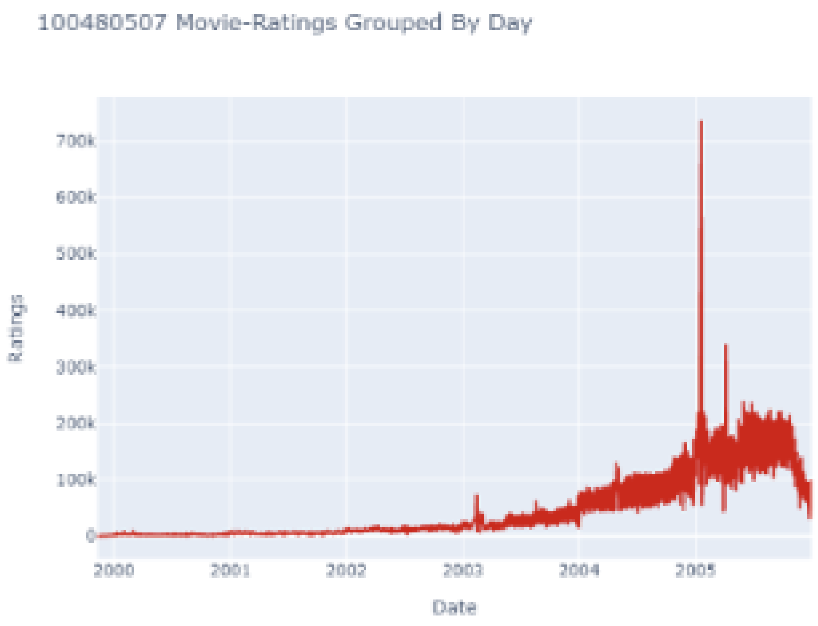

Figure 2-3 Distribution of Movie Rated Date

rather than rating. There is a point worth noticing that low-rated films are commonly considered to be of poor quality.

2.1.3 Movie Rated Date

Netflix had little to no regular ratings between 2000 and 2003. The daily ratings reached their first minor high after 2003. After that, amid variations, the regular ranking numbers begin to indicate an increasing trend. The number of ratings submitted varies over time, with the majority of ratings being submitted in 2005. It is worth recalling that 2005 had two out-of-the-ordinary highs. The distribution of movie rated date is shown in Figure 2-3.

2.1.4 Others

The exponential decay in both the movie and consumer scores is almost flawless. Only a few movies/users have received a significant amount of ratings. See the distribution of the ratings per movie and user in Figure 2-4.

2.2 Data Preprocessing

The training set and test set are separated from the original dataset. After separation, the training set contains 99,072,112 ratings in total for 17,770 movies from 480,189 users. The test set contains 1,408,395 ratings for 16,938 movies from 462,858 movies.

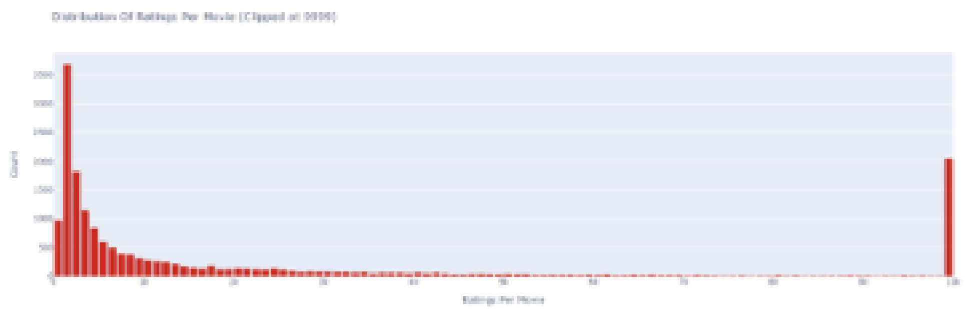

Figure 2-4 Distribution of Ratings Per Movie and User

The columns of rating date will not be taken into consideration. Therefore, that column is deleted.

Different models have different requirements for preprocessing training data, such as dealing with missing values. Therefore, such procedures will be discussed in the following sections, along with the technical details of each model.

3 Modeling Choices

In this section, I will talk about the models I chose for comparison.

3.1 Matrix Factorization with ALS

The reason why Matrix Factorization is chosen and how I de-bias is presented in this section.

3.1.1 Biased Matrix Factorization

Speaking of Recommender Systems, one would think of Matrix Factorization models intuitively. The winning team BellKor of Netflix Prize in 2008, wrote that it is probably the most popular and accurate to use matrix factorization for recommender systems. Compared to neighborhood-based methods like collaborative filtering, matrix factorization provides a more global view [2].

As for matrix factorization, my model aims to factorize the rating matrix R into a matrix U, which is $m \times k$, and a matrix V which is $n \times k$. The rows of U and V are called latent factors, thus the name latent factor models. To explain in more detail, the i^{th} row of U is called the *User Factor*, while the j^{th} row of V is called the *Item Factor*. The ratings are approximated by multiplying matrices U and V.

As I have made the comparison of Matrix Factorization implemented with Gradient Descent (GD), Stochastic Gradient Descent (SGD), and Alternating Least Squares (ALS), ALS has the least loss during training. Therefore, Matrix Factorization is utilized here along with ALS to predict the missing ratings. Here I use the multiplication of U and V to generate final results.

Also, I have taken into account user bias and item bias. Therefore, I will also compare the performance of the model with and without de-biasing.

Different users may have different biases when it comes to rating movies. For example, user 1 may tend to rate movies higher than user 2, which means that even if the trend of their ratings is the same, the average score given by user 1 will be significantly lower than that given by user 2. There also exists the problem of item bias, which is also called popularity bias. For example, a box-office hit tends to have higher ratings. Such an effect is present because people are easily influenced by other people’s ratings online [3].

Biased Matrix Factorization takes each user’ s and each item’ s specific characteristics into consideration, which is the user bias and item bias, denoted in β and γ.

Assume I have a rating matrix whose ratings range from 1 to 5. Associated with each user, I have a variable called user bias β, indicating the general bias of users to rate items. For instance, a generous user is very likely to rate all movies/items highly. In that case, the variable β will be a positive quantity. Similarly, highly popular items will tend to have positive item bias γ. The main change of Biased Matrix Factorization is that the predicted value of the rating is partly explained by $\beta + \gamma$. In consequence, the value is given by the following formula,

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} v_{js} + \beta_i + \gamma_j. \tag{3-1}$$

Thus, the minimization objective function J can be formulated by aggregating the squared errors over the observed entries of the rating matrix as follows,

$$\begin{aligned} J = &\frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \beta_i - \gamma_j - \sum_{s=1}^{k} u_{is} v_{js} \right)^2 \\ &+ \frac{\lambda}{2} \left(\sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 + \sum_{i=1}^{m} \beta_i^2 + \sum_{j=1}^{n} \gamma_j^2 \right). \end{aligned} \tag{3-2}$$

Furthermore, user and item biases can be incorporated into the Alternating Least Squares. Because I add a regularization term, I can hold the variables related to users fixed and solve for the variables related to movies/items. After that, I can fix the movie/item and solve for the other part.

As for the selection of hyperparameters, in Biased Matrix Factorization, the hyperparameters I need to tune include λ for L_2 Regularization, the number of iterations, and the number of latent variables. After cross-validation, I find the most appropriate hyper-parameter combination. In the following part, I will set $\lambda = 0.02, num_iter = 100, k = 100$.

3.1.2 Matrix Factorization with Large Dataset

Intuitively, I wanted to generate a rating matrix to train the model. The rating matrix will be of size $480,189 \times 17,770$. However, as the dataset is relatively large, when trying to generate the rating matrix, pandas cannot handle such a large dataset, and the whole program eventually crashed.

Instead, the first strategy was to divide the training dataset into three parts and generate rating matrices separately. After that, merge the three rating matrices. However, such a method failed in the last merging step with pandas as pd.merge() has to make a copy of each matrix for merging. I then tried a new package called datatable, which claims to perform better with large datasets than pandas. The result is disappointing as well. In the last step, where the datatable will be transformed back to the dataframe, the program crashes.

I had no choice but to consider feeding the algorithm only part of the training matrix each time until all the training data are fed into it. In the end, I divide the training set according to the id of the users.

The problem of how large an interval should be arises. The userid ranges from 1 to $2,649,429$. After considering the trade-off between time and training efficiency, the interval size is set to 100,000, which means there will be 27 intervals in total for training.

3.2 AutoRec

The reason why AutoRec is chosen and some technical details are presented here.

3.2.1 AutoRec Details

Intuitively, collaborative filtering is a good solution for recommender systems as it considers users' information based on their similarity. Latent factor models are the first thing that came to my mind as it is the state-of-the-art model in recommender systems. Other collaborative filtering methods need dimension reduction as a pre-step to provide a more convenient data representation before any estimation is made. Dimension reduction is performed because even an incomplete data matrix can robustly estimate the reduced representation. In terms of latent factor models, dimension reduction and data matrix estimation are completed together in one step [1].

Sedhain et al. implemented collaborative filtering with AutoEncoder, and the model outperforms the state-of-art models in the field of Recommender Systems like Matrix Factorization with Biases. Therefore, I try to build an item-based model and a user-based model with reference to their paper. Based on the intuitions mentioned above, I implemented Matrix Factorization and AutoRec [4].

AutoRec, short for AutoRec: Autoencoders Meet Collaborative Filtering [4], is a neural network-based model in the Recommender Systems. Its main idea is to reduce the dimension by a neural network, whose output is the low dimensional vector with denser information. After obtaining the encoded output, it is fed into the decoder and returns a predicted user/item vector as the final prediction.

The model consists of two parts: encoder and decoder. The encoder compresses the sparse and large user/item vector into dense vectors. On the other side, decoder returns the vector with the same dimensions as the input user/item vector. As for their structure, the encoder is built with a Linear layer and a Sigmoid Layer, whereas the Decoder only contains a Linear layer.

Apart from being used as a model for recommendations, it could also be viewed as a tool for dimension reduction based on the idea of Autoencoder [5]. Furthermore, this could be utilized in further models such as Decision Tree.

3.2.2 AutoRec Strategies with Large Dataset

When handling the Netflix Dataset, I found that it is impossible to generate the whole rating matrix at one time either in Google Colab or my laptop. Thus I found a way to generate a small part of the rating matrix to avoid MemoryError or Runtime Error. Also, to enhance the training speed, GPU is used, and the training tile is significantly shortened.

The rating matrix is partly generated each time with the help of the sparse matrix. Firstly, the number of user/item is determined, depending on the model being user-based or item-based. After determining the number of user/item, it is necessary to reindex their id since the sparse matrix would generate the whole matrix using the row index and column index. Moreover, this step is also applied to the corresponding test data. By feeding the training data in batches to the training process, the required memory is significantly reduced. It could even run on a standard laptop with a relatively high speed. In the meantime, the program uses all the training data as well. When choosing the batch's user/item, there are different strategies as well. Choosing them by index or randomly choosing a certain number of user/item is testified in Google Colab.

In order to train the model faster, GPU is utilized here, and there are a few things needed to be aware of. The number in the batch being fed into the training process should be moderate in case the runtime of cuda is used up. Also, all variables must be in the same device(”cuda”). As a result, the enhancement in training speed is pretty surprising, and it takes about one-fifth of the time trained on the CPU.

3.3 Naïve Bayes

This section will introduce why and how I use the Naive Bayes Model to construct the recommender system.

3.3.1 Naive Bayes Recommendation Systems

I consider the Naive Bayes collaborative filtering algorithm for the dataset because Naive Bayes is best suited for categorical input variables. The ratings in the Netflix Prize dataset are five integers, which can be treated as five categories. Moreover, Naive Bayes is very straightforward and easy to demonstrate, unlike black box models. It also has a low time complexity, which makes it favorable in real-life applications.

Naive Bayes also has its weaknesses. First of all, it assumes that all predictors are independent, which is not always compatible with real-life scenarios and could limit Naive Bayes' usage. However, for the Netflix Prize dataset, there is no evidence suggesting that there exist strong correlations between predictors or features. Another weakness worth noting is that Naive Bayes heavily relies on the calculation of posterior probability. When it cannot calculate a posterior probability for a category, the algorithm automatically fills in with 0. One of the Netflix Prize dataset characteristics is that it is very sparse, as not many users rate the movies. Assigning 0 probability will only result in an over-fitting model. To tackle this problem, a Laplacian smoothing parameter is added to implement Laplacian smoothing.

In the recommendation scenario, the predicted ratings are supposed to be generated based on the observed ratings. Here the possible rating values are denoted in. Hence the ratings are specified in specific ranges. Bayes Rule could be modified as follows,

$$\begin{aligned} &p(r_{ij} = v_s | observed\ ratings\ in\ I_i) \\ =&\frac{p(r_{ij} = v_s)p(observed\ ratings\ in\ I_i | r_{ij} = v_s)}{p(observed\ ratings)}. \end{aligned} \tag{3-3}$$

The denominator could be viewed as a scaling factor, so $p(r_{ij} = v_s observedratingsinI_i) \propto p(r_{ij} = v_s) \times p(observedratingsinI_i r_{ij} = v_s)$. The term $p(r_{ij} = v_s)$, known as the prior probability of rating r_{ij}, is estimated to be the fraction of the user who specifies the rating v_s for item j. However, it is noteworthy that the fraction is only computed on the users who have specified the ratings of item j. For the second term in the right part of the equation, it could be decomposed into the multiplication, based on naïve assumption of conditional independence, which could be expressed in mathematical form as follows,

$$p(observed\ ratings\ in\ I_i | r_{ij} = v_s) = \Pi_{k \in I_i} p(r_{ik} | r_{ij} = v_s). \tag{3-4}$$

The value of $p(r_{ik} | r_{ij} = v_s)$ is estimated by the fraction of users specifying the ratings of item k given the fact that they specify the ratings of the item j to v_s. After having the posterior probability, there are mainly two ways to obtain the final prediction results depending on the output requirement. The first way is to determine the value by the largest value of probability. In this way, the final predictions are discrete values. The second way to estimate the value is to average all rating values with different posterior probabilities. The prediction generated in this way could be continuous instead of discrete.

3.3.2 Naive Bayes with Large Dataset

I consider the Naïve Bayes collaborative filtering algorithm for the dataset because Naive Bayes is best suited for categorical input variables. The ratings in the Netflix Prize dataset are five integers, which can be treated as five categories. Moreover, Naive Bayes can handle the problem of over-fitting and MemoryError. It also achieves better results in accuracy and performance, which makes it favorable in real-life applications.

There is also the problem of over-fitting. The Netflix Prize dataset is very sparse, as not many users rate the movies. Assigning 0 probability will only result in an over-fitting model. To tackle this problem, a Laplacian smoothing parameter is added to implement Laplacian smoothing. Through CrossValidation, I can select an appropriate λ for my model.

Most important is that my model uses the method called distributed computing. Unlike some methods involving the usage of a rating matrix, Naïve Bayes can make predictions with data points' posterior probability after reading in the original dataset, which enables it to consume much less space. However, when running my Naïve Bayes collaborative filtering algorithm on my laptop for the first time, I found that it would take more than five hundred hours to make predictions on the test set. I solved this problem by splitting my test set into a dozen parts and predicting the ratings for each part.

3.4 Neural Collaborative Filtering

I tried to include Neural Collaborative Filtering in my consideration as well.

3.4.1 Neural Collaborative Filtering

I'm considering demonstrating the Neural Collaborative filtering on the Netflix dataset to build a movie recommender system. There are two basic ways of thinking about the recommender system. One is the user and item characterization, another one is feature crosses, in traditional algorithms, the latent factor model starts from embedding the user and item to get the inner product to represent the preference of the users on items. The factorization machines focus on the problem of feature crosses while the AutoRec cannot handle the feature crosses, and the AutoRec model is also weak in dealing with non-linear data. Neural Collaborative filtering is the first deep learning model which can deal with both problems simultaneously, as it can use the linear ability of MF and the non-linear ability of MLP.

The Neural Collaborative filtering network can be understood to be divided into two subnets, one is the Generalized Matrix Factorization, and the other is Multi-Layer Perceptron, both subnets contain the characterized part of user and item, hence here comes out two problems, how to optimize the characterization and how to do feature crosses. For the second problem, I can solve it by using the inner product of the vectors directly. As for the first problem, I need to use the optimization method. For the Generalized Matrix Factorization, the inner product is input to the fully connected layer, and predicted by the SoftMax, compare the optimized object, GMF is almost equal to the LFM. For the MLP part, I first finish loading the trained weight of user and item, then concatenate the vectors, input into the MLP model, and use the non-linear function to activate the process of optimization. During the optimization process, it is needed to split the user and item in the GMF part and MLP part. I can see that the Neural Collaborative filtering model has made some points dealing with both the characterization and feature crosses. Based on this, there can be more methods to optimize the model.

3.4.2 NCF strategies with Large Dataset

Using the NCF model to train the Netflix dataset, I found that it could waste a very long time to train the whole dataset at once as the size is too large. Since then, I decided to split the whole dataset into several documents to many small parts to transform the data into hd5 type, using a dask data frame, I can also use the flow from frame so I can get the cross-entropy loss. About deciding the batch size of the model, I found that if the batch size is big like 512×8, then the result of the cross-entropy loss will be increasing, which is not good. However, if I set the size too small, I cannot deal with this huge dataset. Hence I finally decided to make the batch size 32, so I can get a relatively satisfactory result.

4 Results

Results are presented according to different models.

4.1 Matrix Factorization with ALS

The RMSE score returned by MFALS is 1.2694, and the RMSE score returned by BMFALS is 1.2696. The result of BMFALS is not as good as MFALS in terms of the RMSE of prediction. However, according to Figure 5-1, it is clear that the training loss of BMFALS is much less than MFALS. It is suspected that the unsatisfactory performance of BMFALS may be due to how training data are fed into the algorithm partially each time.

4.2 AutoRec

The RMSE for the item-based model is 0.9851. However, the value loss fluctuated between 1.8 and 2.1. The best RMSE for the user-based model is 1.0851, and the value loss, in the end, is 1.139184.

In my attempts, I split the data obtained into a training set and testing set by 50% to generate the rating matrix as a whole and avoid the problem of memory overflow. However, such a solution is only temporary as I have to deal with the training set, which has more than $90,000,000$ data points and suffers from the memory overflow problem. Therefore, I changed my strategy and generated the rating matrix partially so such problems can be avoided. It has been brought to my attention that after fixing the memory overflow problem, the RMSE has increased compared to the model, which takes in the whole rating matrix. However, the comparison of results between item-based and user-based models is in line with my results.

4.3 Naïve Bayes

Since my time is limited, I only run my user-based Naïve Bayes model on the test set. Just as I have mentioned above, to overcome low computation efficiency, I divide the test set into twelve parts based on the user-id. Afterward, I combine the results corresponding to different parts in the way as follows.

For each part of the test set i, I can calculate its $RMSE_i = \sqrt{\frac{1}{n_i}\sum_{j=1}^{n_i}(\hat{x}_{ij} - x_{ij})^2}$.

Based on this equation, I could further obtain the RMSE for the whole test set, which can be expressed with RMSE corresponding to different parts.

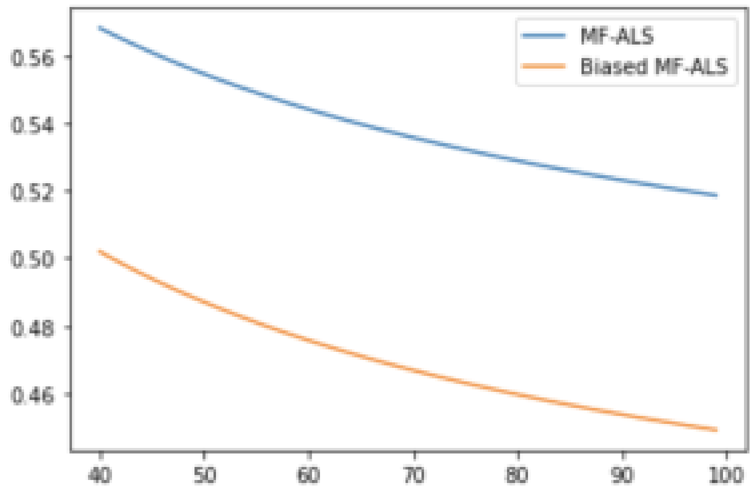

Figure 4-1 Loss History During Training

$$RMSE = \sqrt{\frac{1}{\sum_{i=1}^{12} n_i} \sum_{i=1}^{12} n_i RMSE_i^2}. \tag{4-1}$$

$RMSE_i$ is the RMSE for part i in the test set, and n_i is the number of observations in part i. The RMSE I obtained for the Naïve Bayes model ($\lambda = 0.1$) is 0.9124.

4.4 Neural Collaborative Filtering

As mentioned in the model choices part, as the dataset is too large to be trained at once, I have to split the Netflix dataset into many small parts and set the batch size as small as possible. Using this method, I can get a satisfactory result of the cross-entropy loss. However, when I try to get the RMSE value of the model, this method is not working even though I split the dataset into small parts. Hence I found my method can not work when calculating the RMSE value in a limited time after using the Neural Collaborative Filtering structure building a recommender system.

5 Discussions

In this section, I discussed the problems faced by all the meth the problems and some technical details regarding different models.

5.1 General Problems and Proposed Possible Solution

Two problems exist in all four models: the dataset is too large, and that training and predicting take too much time.

As I've mentioned above, the dataset of the Netflix Prize dataset has more than $100,000,000$ data points, and the file size is almost 3 GB, which is what is called a medium-sized dataset for machine learning tasks. However, such a size puts the dataset in an awkward position. It will cause RAM overflow if processed on a regular PC but is not so large that it is necessary to create a platform on the cloud with scalable and parallelized machine learning techniques.

The test set and train set are constructed by taking the probe set out of the original dataset. As a result, the train set has $99,072,112$ ratings, where $480,189$ users rated $17,770$ movies, and the test set has $1,408,395$ ratings where $480,189$ users rated $16,938$ movies. Reading the CSV files containing these data into data frames is not a big problem, but the problem arises when generating the rating matrix. The rating matrix is generated differently for item-based, and userbased models as the rows and columns might switch. In general, the dimension of the rating matrix could be as large as $480,189 \times 17,770$ or $17,770 \times 480,189$, which often causes the RAM of my computer to overflow.

As for the time problem, since the dataset is large, the training could take up to 30 hours for the models implemented with matrix factorization. However, the situation is different for the Naive Bayes model. The problem of RAM crashing is also present in this model. To avoid such trouble, my Naive Bayes Model does not need to generate the rating matrix and stores the data in three columns: user_id, movie_id, and ratings. The training is done by calculating the prior probabilities using the data in the training dataset. My model integrates training and prediction. If a system only has to generate recommendations for a few users, my model is all right. However, it is not realistic for the Netflix Prize problem as my model takes a time as long as 720 hours to generate predictions for the test set. Although my first attempt failed, it still provided a valuable lesson. In the end, I transferred the solution for the matrix factorization model training problem to the Naive Bayes model by splitting the training data.

I have come up with some possible solutions. As mentioned before, the biggest problem I found is the big dataset. This problem has been a tough barrier when trying to fit the Netflix

Dataset to some comparatively complicated models like the Random forest. Hence I think I can improve the models by handling the big dataset better. So I might consider dividing the dataset into small sizes and solving the dataset by parts first. If that does not work well, I can improve the speed of machine learning by running dask. Hence I may consider using the cluster manager, Kubernetes in the future to do cloud computing getting the model result.

To reduce the big size of the dataset, I'm also considering the method of Principal Component Analysis. Using PCA could help me reduce the dimensionality of the huge rating matrix to pick up the mean features hence reducing the complexity of the later modeling. However, how to use the method of dimensionality reduction on the recommender system still needs more research to find out if that could work.

For the AutoRec model, the result of RMSE is not good enough. This is because the model has not dealt with the missing values in the dataset. So I tried to fill the missing values with 2.5 and update the values every round. However, this does not work well. Therefore, I might consider improving the model to Collaborative Denoising Auto-Encoder (CDAE) in the future, which considers the user's features. Model CDAE can use maskout or dropout to apply random noise hence having better robustness than AutoRec.

Other than the considerations above, I think I may also add the features of the releasing date of the movies and the rating dates of the users to compare the similarity of users and items when doing collaborative filtering to improve the accuracy of the predicting result. Moreover, I can also add the name of the movie to be a potential feature as the movies of the same series will probably be affected by the earlier ones in the series, and I believe adding the consideration of these features in the future could help to make a better recommender system with a lower RMSE.

5.2 Matrix Factorization with ALS

The main problem of the implementation of Matrix Factorization is how to generate the rating matrix with such an extensive dataset.

Usually, recommender systems have to deal with high dimensional rating matrices. The key to matrix factorization is that the method can decompose the rating matrix into a user matrix and an item matrix, thus immensely reducing dimensionality. However, as mentioned earlier, the dataset is so large that turning it into a rating matrix is difficult due to my hardware's limitation (the computer without enough RAM) and the software (pandas do not perform well with such a large dataset).

My solution is to feed the model partial data each time until all the training data is fed into the algorithm. Indeed, in this way, it becomes possible for the program to run without crashing. However, it affects the performance of the model as a whole, especially its predictive power. I suspect the reason why RMSE scores for the Matrix Factorization model are higher than other models is that the rating matrix is sliced.

The problem of how to slice the rating matrix rises naturally. For my model, the interval is chosen to be $100,000$. The larger the interval is, the more information is preserved and processed during training. Therefore, I want to keep the interval as large as possible. After trying out $1,000,000$, $500,000$, $250,000$ and $100,000$, an interval of $100,000$ is the most suitable as the running speed is appropriate and the program will not crash.

5.3 AutoRec

There are a few things needed to be aware of when training the model. Due to the sparsity of the rating matrix, the input matrix should be firstly imputed by assigning the missing values to 0. When reading the data into a notebook, MemoryError may occur because the room for storing the data exceeds the RAM of my device. So here I avoid using NumPy and pandas. Instead, Scipy and torch.Tensors are used here to save memory thanks to the data structure of the sparse matrix built inside the packages. Also, it is critical to free the memory after finishing using the corresponding variables. Unlike ALS updating the missing value each epoch, AutoRec keeps the zeros and computes the loss upon the observed ratings. Thus when calling the loss.backward() method in Pytorch, the corresponding model's parameters are updated through the dynamic graph while other parameters stay still. In addition, to constrain the parameters' value, L_2 normalization is used when computing the loss and gradient to carry out backpropagation. Nevertheless, it is noteworthy that the L_2 norm only penalizes the w instead of w and b (see definition below). Otherwise, it will make the loss much larger. Below is the mathematical expression of the encoder and decoder.

When using the model to predict the ratings, the missing values of the input matrix should also be assigned as zeros. Moreover, the output of the decoder is the predicted ratings for each user and item. From the discussions above, there are a few things and parameters worthy of being discussed here.

First, item-based and user-based models' performances are compared. I implemented different models of user-based and item-based types. The user-based model encodes the user's rating vector into a lower dimension, while the item-based model encodes the item's rating matrix. After exploring the dataset, it is found that the item vector is denser than the user vector, indicating that the item vector contains more information, thus may lead to better performance. To test the model, I split the dataset into a train and validation part. Since the model is an encoder-decoder structure, it captures the features of the input vectors by the encoder. It then predicts the missing values by the decoder, which is not related to the overall rating matrix. After training the model, it is found that when holding other hyper-parameters (learning rate and L_2 regularization factor, etc.) equal, the item-based model has a lower training loss

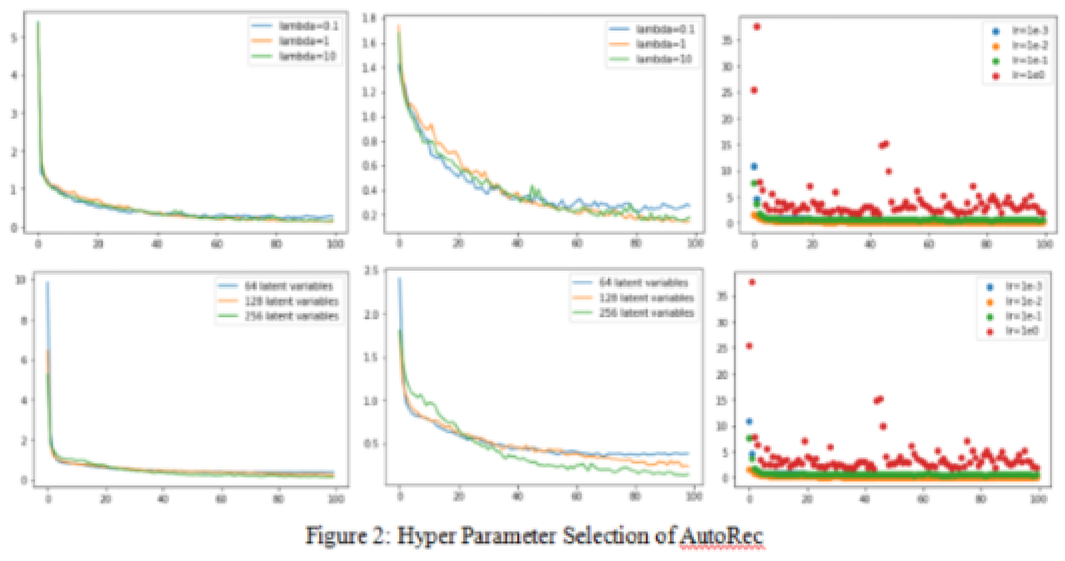

Figure 5-1 Hyperparameters Selection of AutoRec

value, which is 0.00771, than that of the user-based model, which has a loss of 0.13494 on the validation dataset.

Having discussed the performances of the two models, the choice of hyper-parameter will be discussed as well. As for the hyper-parameters, including learning rate, Lambda (penalty of L_2 Regularization), dimension of latent variables, I use the same strategy as in Matrix Factorization. Below is a detailed analysis of the results. Due to the size of the Netflix Dataset, the computational constraints of Google Colab, and the limitations of my laptop, it is difficult for me to design and train a model this complicated on the Netflix Prize dataset in merely seven days. Therefore, to better understand the influence of these hyper-parameters so that I can conduct a comprehensive analysis of the model, I utilize AutoRec on a toy dataset to figure out the strategies of hyper-parameters selection.

Another problem concerning hyper-parameter selection is how to choose the learning rate. The value of the learning rate influences the speed of convergence and final RMSE as well. By assigning a learning rate from 1e-3 to 1e-1, I found that a more significant learning rate would lead to a faster drop in the learning curve. Nevertheless, a lower learning rate could converge the loss to a smaller value, thus improving the model's performance. The appropriate learning rate after testing is 0.001.

There is another hyperparameter Lambda that is to be taken into consideration. The term L_2 Regularization is used to prevent the model from over-fitting. The appropriate value of Lambda is 1. The way to determine the value of the regularization term is similar to the latent dimensionality above, where multiple possible values including 0.01, 0.1, 1, 10 are tested, and their performances are compared. As for the final choice, the value of 1 is used in the regularization term to avoid the problem of over-fitting.

The selection of the dimension of latent variables reflects the compression of information when doing dimension reduction. Less dimension for latent variables indicates more loss of information during this procedure, which may harm the RMSE value. The result accords well with theoretical analysis. The latent dimensionality is determined by multiple factors, such as the RAM of the computer and memory of the GPU, etc. I tested the model's performance with different dimensionality, including 128, 256, 512, 1024, 2048, and compared their performances in RMSE values, as shown in Figure 5-1. Moreover, after the comparison, it turned out that models with higher dimensionality achieve better performance with less RMSE value. More information being preserved by higher-dimensional representation may contribute to the final result. However, larger dimensionality leads to a heavier burden for computation and longer training time. So I utilized 1024 as the final latent dimensionality.

5.4 Naïve Bayes

As I have mentioned above, Naïve Bayes can tackle the MemoryError by only using related rating records in the original data in the forms of pd.DataFrame. For example, to calculate the prior of, I need to count the number of users who have specified the ratings of item j to be and the number of users who have specified the ratings of item j. Thus, these numbers are counted from the original dataset by using np.where to filter the useful information.

It should be noted that I only store the training set without the likelihood parameters in my model. Otherwise, it is also inevitable to encounter the MemoryError when training the Naïve Bayes model. However, it is a trade-off between time and space. Reducing the use of memory will lead to the rise in the time for predicting. To solve low computation efficiency, I divide the test data into several parts to obtain the RMSE on the test set.

As for the selection of hyper-parameters in the Naive Bayes model, I am mainly concerned with choosing the smoothing laplacian parameter. In the Naïve Bayes model implemented, there is only one hyper-parameter needed to be tuned here. It has come to my attention that although it is agreed that the smoothing parameter can reach good performance at 1, when implemented in my model, the problem is that the denominator could be large after adding the smoothing parameter. Simultaneously, the numerator remains relatively small, which results in the probability being close to 0. Due to the huge amount of data, I conduct the cross-validation on a minitraining set generated by the stratification sampling based on user-id. I testified different values from 0.01 to 1 to see how it influences the final result, and below is the graph illustrating its influence.

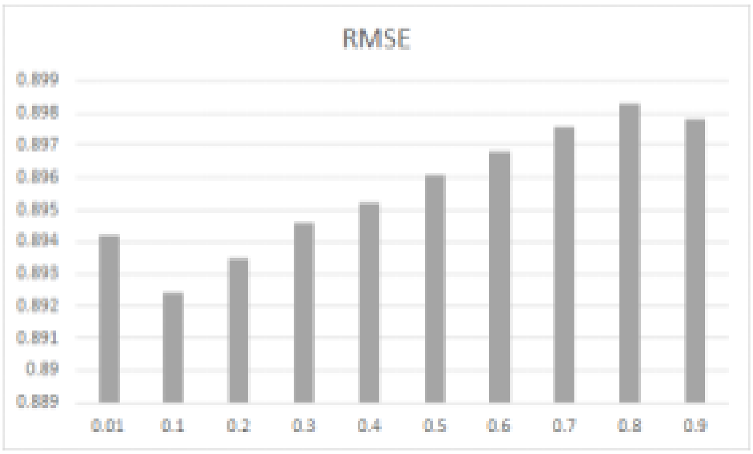

Figure 5-2 Selection of Hyperparameter in Naïve Bayes Model

5.5 Neural Collaborative Filtering

There are two cases where I use the NCF structure by now, GMF and MLP, one for linear and one for non-linear, hence I can mix these two together. To ensure the mixed model having higher flexibility, I allow the GMF and the MLP to learn to embed independently and link their final outputs of the hidden layer. I use ReLU to activate the MLP layer, combine the linear feature of the MF and the non-linear feature of the DNNs to build the potential structure between the user and item. And this way of combination is called Neural Matrix Factorization, as every parameter in this model can be calculated by backpropagation.

As the function in Neural Matrix Factorization has nonconvexity, the optimization method based on steps can only achieve the local optimization. Hence the initialization made big attribution in the convergence and the efficiency of the deep learning model. Hence I should pre-train the GMF and the MLP to initialize the Neural Matrix Factorization. To achieve this, I should use adaptive moment estimation to suit the speed of learning of every parameter within frequently updating with an unoften change in parameters. This method works faster than the method of normal SGD and solves the problem of changing learning frequency.

This third model Neural Matrix Factorization using the structure of Neural Collaborative Filtering is not complicated and should be able to be used in many other systems. It is meaningful in the deep learning recommender system and opened a new way of the system. In the future, I can try to apply the Neural Collaborative filtering model in paired learning, expand the support information of building models.

6 Conclusion and Future Implications

The Matrix Factorization model has the worst performance compared to AutoRec and Naive Bayes models, whether the model is implemented with or without bias.

As a Neural Network based method, AutoEncoder is initially used in dimension reduction, in which process the sparse information is condensed and can be retrieved by the decoder part of the model. Based on this idea, it is utilized in the Recommender Systems to capture the characteristics of the user's preferences or the movie's attractions for different people. And it is especially useful and convenient when it comes to a large dataset by feeding the input batches to batches rather than generating the complete rating matrix. However, due to the limitation of its structure, the model couldn't handle the missing value efficiently and accurately. When training the model, the backward process is only implemented on the observed ratings, which means that the less popular movies may not receive enough data for training. Overall, it achieves satisfying performance and is reasonable in the scenario of the recommender system.

As for the Naive Bayes Model, it treats ratings in the Netflix dataset as a categorical variable and applies Bayes rules to calculate the posterior probability, which is later used for predictions. Compared with Matrix Factorization, there is no need to generate a huge rating matrix when training the machine learning model; thus, it could prevent the problem of MemoryError effectively. When it comes to computation efficiency, I split the test set into 12 parts, combine the RMSE corresponding to each part and finally obtain the RMSE for the Naive Bayes. Last but not least, Naive Bayes can also handle the problem caused by a large number of missing values in the dataset. By tuning the Laplacian smoothing parameter on the validation set, I'm able to select an appropriate hyperparameter for our model, and it outperforms both Biased Matrix Factorization with ALS and AutoRec.

References

[1] Aggarwal C C, et al. Recommender systems: volume 1[M]. Springer, 2016.

[2] Bell R M, Koren Y, Volinsky C. The bellkor 2008 solution to the netflix prize[J]. Statistics Research Department at AT&T Research, 2008, 1(1).

[3] Adomavicius G, Bockstedt J, Curley S, et al. De-biasing user preference ratings in recommender systems[C]//RecSys 2014 Workshop on Interfaces and Human Decision Making for Recommender Systems (IntRS 2014). 2014: 2-9.

[4] Sedhain S, Menon A K, Sanner S, et al. Autorec: Autoencoders meet collaborative filtering [C]//Proceedings of the 24th international conference on World Wide Web. 2015: 111-112.

[5] Rumelhart D E, Hinton G E, Williams R J. Learning internal representations by error propagation[R]. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.