Complete End to End study of Recommendation System

Sarveshkumar Purohit
MTech AI, IITH
ai22mtech14006@iith.ac.in

Soumyanetra Pal
MTech AI, IITH
ai22mtech14005@iith.ac.in

Guided By:
Prof. Shantanu Desai
Department of AI,IITH

ABSTRACT **The Netflix Prize was a competition launched by the online streaming company Netflix in 2006, offering a grand prize of 1 million dollar to any individual or team that could improve the accuracy of its movie recommendation algorithm by at least 10%. A team of computer engineers from AT&T Labs won the grand prize by reducing the RMSE loss from 0.9525 (Loss reported by Netfix's Recommender Algorithm called CineMatch) to 0.8572. As the Netflix Prize competition has demonstrated, matrix factorization models(eg. SVD) are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels.**
In this project, we aim to study the matrix factorization techniques like SVD and SVD++ used in the paper published by the winning team of Netfilx Prize challenge from AT&T Labs in the IEEE Computer Society [1]. The Part of the machine learning algorithm suggested by winning team in this paper is still used by Netflix in production server.
This Report is made as part of the course project of EP4130/PH6130 (offered at IITH during Spring 2023), Code is available at Github or Google Collab Link

INDEX TERMS Matrix Factorization Techniques: Singular Value Decomposition (SVD) and SVD++ for Recommender Systems, User-User and Item-Item collaborative filtering, XGBoost, SurPRISE library

I. INTRODUCTION

Good personalized recommendations can add another dimension to the user experience. E-commerce leaders like Amazon.com and Netflix have made recommender systems a salient part of their websites.

Recommender systems provide users with personalized suggestions for products or services. These systems often rely on collaborating filtering (CF), where past transactions are analyzed in order to establish connections between users and products. The most common approach to CF is based on neighborhood models, which originate from similarities between products or users.

Our attempt is to understand collaborating filtering (CF) techniques which uses matrix factorization as core.

II. BACKGROUND AND RELATED WORK

A. RECOMMENDATION PROBLEM

Recommendation problems in machine learning involve predicting the preferences or interests of a user based on their historical behavior and other data. This type of problem is common in e-commerce, where companies use recommendation algorithms to suggest products to users based on their browsing and purchase history.

B. COLD START PROBLEM

The cold-start problem [2] is a common challenge in recommendation problems that occurs when there is insufficient data about a user or an item to make accurate recommendations. In other words, the system has no or very little information about the user's preferences or the item's attributes, which makes it difficult to provide relevant recommendations. We will use surprise [3] library to tackle this problem.

C. SPARSE MATRIX AS A DATA STRUCTURE OF RECOMMENDATION SYSTEM

To put Recommendation System as Regression Problem, we need the data provided in the following format-

$$D = \begin{array}{c} \\ u_1 \\ u_2 \\ \dots \\ u_p \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & \dots & m_q \end{array} \\ \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1q} \\ r_{21} & r_{22} & r_{23} & \dots & r_{2q} \\ \dots & \dots & \dots & \dots & \dots \\ r_{p1} & r_{p2} & r_{p3} & \dots & r_{pq} \end{pmatrix}_{p \times q} \end{array} \tag{1}$$

where,

- u_i are the users id
- m_j are movies id
- r_{ij} is the rating given to m_j by u_i. r_{ij} can be any real value in a range,

The main challenge in recommendation problems is dealing with the sparsity of the data. In most cases, users only interact with a small fraction of the available items, which makes it difficult to learn their preferences for the rest of the items. We will later use Compressed Sparse Row (CSR) [4], [5] representation of sparse matrix for efficient storage.

D. COLLABORATIVE FILTERING

Collaborative filtering is a commonly used technique in recommendation systems. **It is a process of predicting a user's interests or preferences by analyzing the past behavior of other users who have similar tastes.** In other words, collaborative filtering is based on the idea that people who agree in their assessments of certain items in the past are likely to agree again in the future. This term was coined in Using Collaborative Filtering to Weave an Information Tapestry [6]. Since then it managed to get huge attention.

1) User-User Similarity

'Users who have agreed in the past tends to agree in the future'. With this idea, and the data of the structure described in Table 1, we aimed to find similar users. Consider each row of Table 1 to be the vector representation of each user. Using the vector representations, we can now measure the similarity between any two users. Taking **Cosine Similarity** [7] as our metric for Similarity, the similarity between two users u_i and u_j, sim(u_i, u_j) is given by-

$$sim(u_i, u_j) = \frac{u_i^T \cdot u_j}{\|u_i\|_2 \|u_j\|_2} \tag{2}$$

The same similarity can also be calculated using **Pearson Correlation Coefficient** given by-

$$sim(u_i, u_j) = \frac{\sum_{k=1}^{n}(u_{ik} - \overline{u_i})(u_{jk} - \overline{u_j})}{\sqrt{\sum_{k=1}^{n}(u_{ik} - \overline{u_i})^2 (u_{jk} - \overline{u_j})^2}} \tag{3}$$

Such similarities between every user will give us an **User-User Similarity Matrix** S^u, where each entry is given by-

$$S_{ij}^u = sim(u_i, u_j) \tag{4}$$

Recommending movies to users now is easy. Suppose we want to recommend movies to user-5(u_5). Find 'k' most similar users to u_5 and recommend the movies liked by those 'k' users.

However, there is one issue with this approach. User's likes change over time and this is hard to capture by the above technique. We can try to maybe consider last few months data to predict but that does not solve the problem entirely. This lays the motivation for the following method.

2) item-item Similarity

'Similar items are rated in a similar ways'. With this idea, and the data of the structure described in equation1, let's aim to find similar items. Consider each column of equation1 to be the vector representation of each item. Using the vector representations, we can now measure the similarity between any two movies. Taking **Cosine Similarity** [7] as our metric for Similarity, the similarity between two users m_i and m_j, sim(m_i, m_j) is given by-

$$sim(m_i, m_j) = \frac{m_i^T \cdot m_j}{\|m_i\|_2 \|m_j\|_2} \tag{5}$$

The same similarity can also be calculated using Pearson Correlation Coefficient given by-

$$sim(m_i, m_j) = \frac{\sum_{k=1}^{n}(m_{ik} - \overline{m_i})(m_{jk} - \overline{m_j})}{\sqrt{\sum_{k=1}^{n}(m_{ik} - \overline{m_i})^2 (m_{jk} - \overline{m_j})^2}} \tag{6}$$

Such similarities between every user will give us an Movie-Movie Similarity Matrix S^m, where each entry is given by-

$$S_{ij}^m = sim(m_i, m_j) \tag{7}$$

Recommending movies to users now is easy. Suppose we want to recommend movies to user-5(u_5). List the movies liked by the user and find most similar movies to them. Recommend the top movies from them .

It is also observed that the ratings of the movies does not fluctuate much after a few initial days, so it is independent of the temporal dependency unlike the previous approach.

E. MATRIX FACTORIZATION

1) SVD

Singular Value Decomposition is a popular Matrix Factorization technique used in Data Analysis. Given a matrix D in equation - x, SVD factorizes it into three matrices:

$$\mathbf{D}_{p\times q} = \mathbf{U}_{p\times p} \times \mathbf{\Sigma}_{p\times q} \times \mathbf{V}^{\mathrm{T}}_{q\times q} \tag{8}$$

where,

- U and V are orthonormal matrices
- Σ is a diagonal matrix with signular values as its entries.

2) SVD++

SVD++ [8] is an extension of SVD, where both explicit and implicit feedbacks are taken into consideration. Explicit feedbacks are those when an user watches a movie and gives a rating explicitly. However, often times, users watch a movie and doesn't rate it. Sometimes they don't complete watching a movie. These **latent informations** are never explicitly available to us but are potentially very important to make correct recommendations. SVD++ comes as a handy technique to deal with it.

F. XGBOOST LIBRARY

XGBoost [9] (eXtreme Gradient Boosting) is a popular open-source software library for gradient boosting algorithms [10].

It is designed to be scalable and efficient, making it a powerful tool for solving a wide range of machine learning problems. One of the key advantages of XGBoost is its ability to handle large datasets with a large number of features. It uses a distributed computing model, which allows it to process data in parallel on multiple cores or nodes. XGBoost also includes several advanced features, such as regularization, parallel processing, and handling missing data.

G. SURPRISE LIBRARY

SurPRISE [3] (roughly) stands for Simple Python RecommendatIon System Engine Surprise is a Python scikit building and analyzing recommender systems. It provides a simple interface for loading data, configuring recommendation algorithms, and evaluating model performance.

Surprise offers a range of built-in algorithms, including basic collaborative filtering methods such as k-NN and matrix factorization techniques such as SVD and NMF. It also supports more advanced techniques such as SVD++, which incorporates implicit feedback, which can handle very large datasets efficiently.

In addition, Surprise provides a range of evaluation metrics for assessing the performance of recommendation models, including RMSE, MAE, and FCP. It also supports cross-validation and grid search to help optimize model parameters.

Overall, Surprise is a powerful and easy-to-use library that can help you quickly build and evaluate recommendation systems.Later we will show use how we have used Surprise library for our study of recommendation system.

III. DATASET

We have used the official dataset provided by Netflix for the Netflix prize challenge. [11]

Feature name	number of entries
Total no of ratings	100480507
Total No of Users	480189
Total No of movies	17770

TABLE 1: Statistics of the dataset

For our training and testing data, we used an 80:20 split after sorting the data in ascending order of the date on which the rating was given. We did not shuffle the data after sorting since we wanted to retain the temporal characteristic of the dataset. This was because we aimed to train our model to predict a user's future movie preferences based on their movie-watching history.

IV. OUR IMPLEMENTATION APPROACH

In this section, We will first provide an walk-through of Exploratory data analysis (EDA)

A. EXPLORATORY DATA ANALYSIS (EDA)

Here we have explored the dataset to analyse certain latent statistical features. This will immensely help in choosing and implementing models.

1) Histrogram of the Distribution of Ratings over Training dataset shows that users rate a movie 4 more often than 1, 2, 3 or 5. Figure1

FIGURE 1: Histogram of movie rating

2) The distribution of Ratings with respect to months of each year beautifully draws the year on year evolution of Netflix. It shows that Netflix has grown rapidly after 2004, which also makes sense because prior to 2004 there was hardly any access to Internet. Figure2 graphically describes the trend.

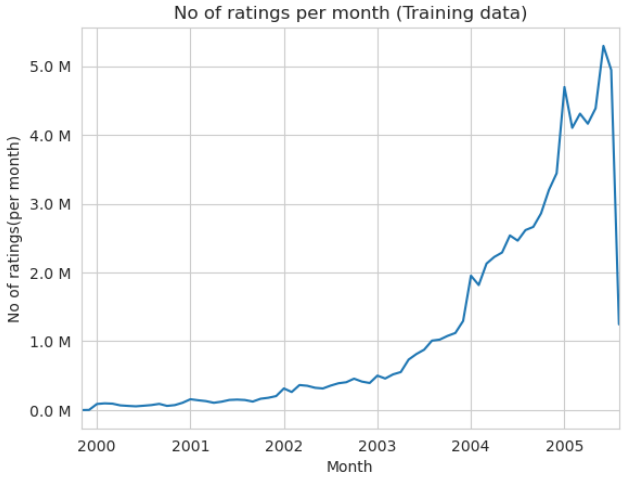

FIGURE 2: Distribution of ratings per month

3) Experiments were made by adding new features to the original dataset. Adding the 'Weekday' feature to the dataset gave us an interesting insight. Asked, which day of the week should see the maximum number of movies watched and rated? Weekends, isn't it?... well the data says 'Tuesday'. Figure3 shows the histogram of number of ratings with respect to week days.
4) The above skewness naturally inclines us to think that there might be a similar trend on user rating movies in a particular way at particular days of the week.

However, that isn't the case. The boxplot in Figure4 clearly shows an uniform distribution of ratings on each day of the week, meaning that users rate a movie based on its merit - not on the day of the week he/she watches it.

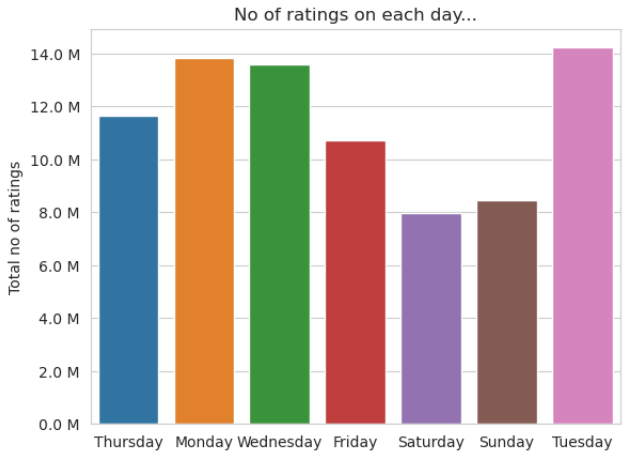

FIGURE 3: Histogram of rating per weekday

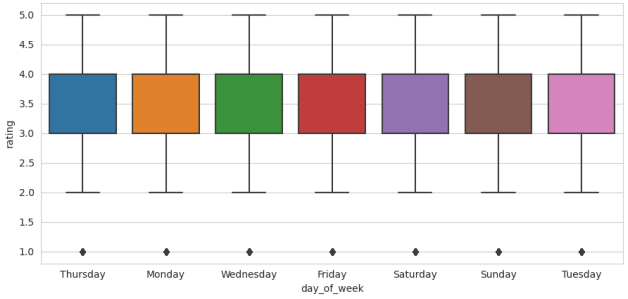

FIGURE 4: Box plot for the rating value given on each weekday

5) Quantiles bring forth interesting insights. From Figure5, we can infer that atmost 89 movies are rated by 50% of the users while the average rating per user is approximately 198 movies! Such a big difference is also explainable, Quantiles to the rescue. On carefully observing the plot in Figure5, we see a sudden rise depicting only 0.05% of the user have rated between 749 to as many as 17112 movies!
6) Since we have a sparse data, a smart representation of it compared to conventional matrix representation will help us work with it better. We have used the CSR matrix representation under the scipy library [5] to effectively represent our data. This representation helps us calculate user-user and item-item similarities later

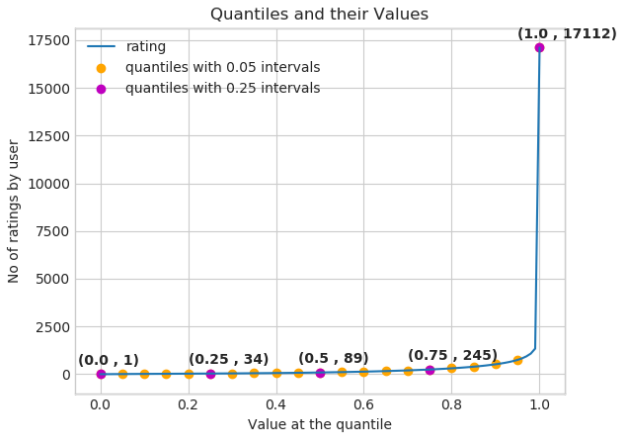

FIGURE 5: quantiles

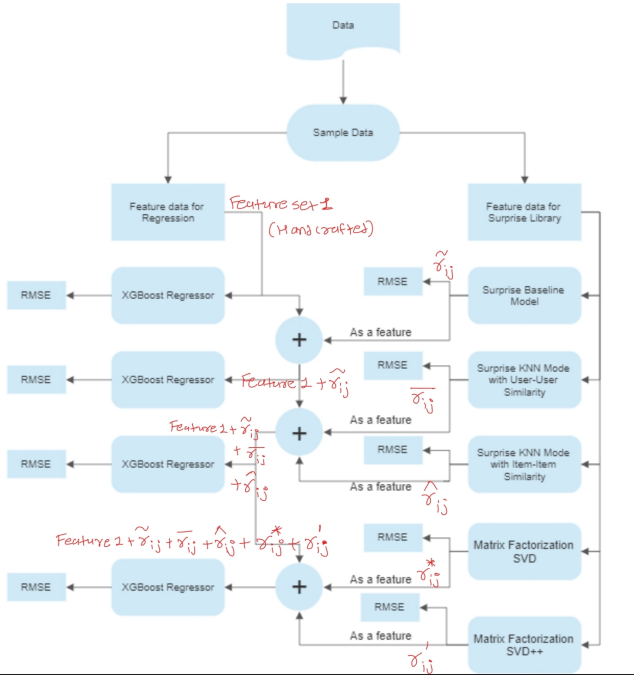

FIGURE 6: Overview of the modelling strategy

B. OVERVIEW OF THE MODELLING STRATEGY.

Because of the hardware constrain we are sampling 9000 users and 1200 movies from the train data and 4000 users and 800 movies from the test data.

1) Handcrafted Features

From the CSR Sparse Representation [5], we have created 13 handcrafted features (Feature Set - 1, in Figure6). This will be the input to XgBoost Regressor - 1. Given an user u_i and a movie m_j, following are the 13 handcrafted features:

- **GAvg**: Average rating of all the movie ratings.
- **UAvg**: Average rating of user(u_i). Encodes how critical u_i is!

- **MAvg**: Average rating of movie(m_j). Encodes how popular m_j is!
- **Similar users rating of this movie**(m_j): By using the user-user similarity we will find the list of the users who are similar to the user (u_i), from this list we will select top 5 user who has watched the movie (m_j), and the rating given by these selected similar user to movie(m_j) will become our features(sur1, sur2, sur3, sur4, sur5).
- **Similar movies rated by this user**(u_i): By using the movie-movie similarity we will find the list of the similar movies watched by the user (u_i), from this list, ratings for top 5 similar movies, will become our features(smr1, smr2, smr3, smr4, smr5).

2) XgBoost Regression Using 13 features (Xg_13)

Data created with the 13 handcrafted featrues are used to train a XgBoost Model. Figure7 shows how important a role did each of these feature play towards giving the predicted rating for an user u_i.

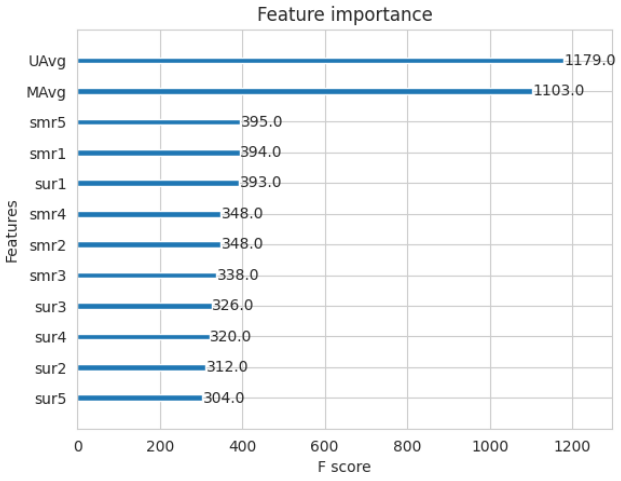

FIGURE 7: Feature score for XgBoost Regression trained Using 13 features

3) Surprise Baseline Model

In order to use the Surprise library's recommendation model, we need to convert our dataset into a format that is compatible with the library. The library provides a convenient way to do this through the 'Reader' and 'Dataset' classes.First, we have created a 'Reader' object, which specifies the format of our input data.Then, we have used the reader object to load our dataset into a 'Dataset' object, which can be used to train and test the recommendation model.

In Surprise, the baseline model is a simple model that predicts the rating for each item and user. The predicted rating for a given user-item pair is calculated as follows:

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i \tag{9}$$

where,

- $\boldsymbol{\mu}$: Average of all trainings in training data.
- $\boldsymbol{b_u}$: User bias
- $\boldsymbol{b_i}$: Item bias (movie biases)

Surprise provides a 'BaselineOnly' class that implements this baseline model. We have created an instance of 'BaselineOnly' and fit it to our data using the fit() method. It use the MSE loss with L2 regulariser as the optimizing function. The expression for the same is given by-

$$\arg\min_{b_u,b_i} \sum_{r_{ui}\in R_{train}} \left((r_{ui} - (\mu + b_u + b_i)\right)^2 + \lambda \left(b_u^2 + b_i^2\right)). \tag{10}$$

The prediction generated by Baseline model, we will refer it as $\tilde{r}_{uj}$ will be use as feature for XgBoost Regression Using 14 features.

4) XgBoost Regression Using 14 features (Xg_14)

Now that we have a predicted rating for a given u_i from the baseline model, we can use that as a featrue along with the 13 featrues i.e Feature set-1 + $\tilde{r}_{uj}$ to train a XgBoost model.

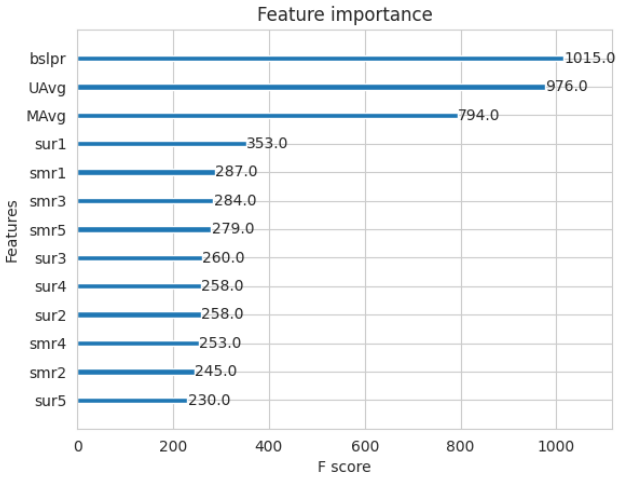

FIGURE 8: Feature score for XgBoost Regression trained Using 14 features

5) Surprise KNN Model using user-user similarity (KNN_uu)

While creating handcrafted features (sur1, sur2, sur3, sur4, sur5) we have only focused on 5 similar(nearest) users. Thinking on similar lines, Surprise library has this awesome implementation of looking at K nearest neighbours(users), the KNN model, to predict the ratings. The expression for the same is given by -

$$\bar{r}_{ui} = b_{ui} + \frac{\sum\limits_{v\in N_i^k(u)} \text{sim}(u,v)\cdot(r_{vi} - b_{vi})}{\sum\limits_{v\in N_i^k(u)} \text{sim}(u,v)} \tag{11}$$

where,

- $\boldsymbol{b_{ui}}$: Baseline prediction of (user,movie) rating
- $\boldsymbol{N_i^k(u)}$: Set of K similar users (neighbours) of user u_i who rated movie m_j

- $\boldsymbol{sim(u,v)}$: Similarity between users u and v Generally, cosine similarity(equation 2) or Pearson correlation coefficient(equation 3) is used. But Surprise uses shrunk Pearson-baseline correlation coefficient, which is based on the pearson Baseline similarity.

6) Surprise KNN Model using movie-movie similarity (KNN_mm)

While creating handcrafted features (smr1, smr2, smr3, smr4, smr5) we have only focused on 5 similar(nearest) movies. Thinking on similar lines, Surprise library has this awesome implementation of looking at K nearest neighbours(movies), the KNN model, to predict the ratings. The expression for the same is given by -

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j) \cdot (r_{uj} - b_{uj})}{\sum\limits_{j \in N_u^k(j)} \text{sim}(i,j)} \tag{12}$$

where,

- $\boldsymbol{b_{ui}}$: Baseline prediction of (user,movie) rating
- $\boldsymbol{N_u^k(i)}$: Set of K similar movies (neighbours) of user u_i who rated movie m_j
- $\boldsymbol{sim(i,j)}$: Similarity between movies i and j Generally, cosine similarity(equation 2) or Pearson correlation coefficient(equation 5) is used. But we use shrunk Pearson-baseline correlation coefficient, which is based on the pearson Baseline similarity.

7) XgBoost Regression Using 16 features (Xg_16)

Following same method as in XgBoost Regression Using 14 features (Xg_14), this time we will consider the predicted ratings from above KNN models(Both user-user and movie-movie) as features, along with the 13 handcrafted featrues i.e Feature set-1 + $\tilde{r}_{ui} + \bar{r}_{ui} + \hat{r}_{ui}$ to train a XgBoost model.

8) Matrix Factorization (SVD)

The famous SVD algorithm, as popularized by Simon Fite [12] during the Netflix Prize is,

$$r^*_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{13}$$

- $\boldsymbol{q_i}$ - Representation of item(movie) in latent factor space
- $\boldsymbol{p_u}$ - Representation of user in new latent factor space

Optimization problem with user item interactions and regularization (to avoid overfitting)

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - r^*_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right) \tag{14}$$

The minimization is performed by a very straightforward stochastic gradient descent, The predicted result r^*_{ui} will be again used later to train XgBoost Regression Using 18 features

9) Matrix Factorization (SVD++)

The SVD++ algorithm, an extension of SVD taking into account implicit ratings.

$$r^{'}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\right) \tag{15}$$

- $\boldsymbol{I_u}$ - the set of all items rated by user u
- $\boldsymbol{y_j}$ - Our new set of item factors that capture implicit ratings.

Optimization problem with user item interactions and regularization (to avoid overfitting)

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - r^{'}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 + ||y_j||^2\right) \tag{16}$$

The minimization is performed by a very straightforward stochastic gradient descent, The predicted result $r^{'}_{ui}$ will be again used later to train XgBoost Regression Using 18 features

10) XgBoost Regression Using 18 features (Xg_18)

This time we will consider the predicted ratings from SVD and SVD++ as features, along with the KNN predictions and the 13 handcrafted featrues i.e Feature set-1 + $\tilde{r}_{ui}$ + $\bar{r}_{ui}$+ $\hat{r}_{ui}$ + r^*_{ui} + $r^{'}_{ui}$ to train a XgBoost model.

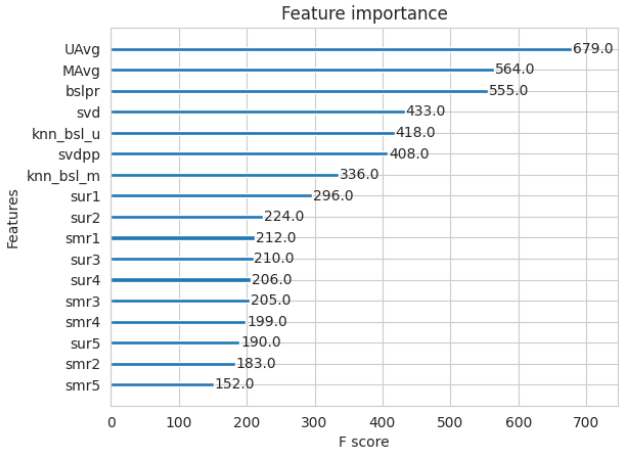

FIGURE 9: Feature score for XgBoost Regression trained Using 18 features

V. EXPERIMENTS AND RESULTS

The following table is a comparative study of how our models performed with different features-

Model	RMSE
Xg_13	1.14690
Baseline	1.06970
Xg_14	1.12258
KNN_uu	1.06974
KNN_mm	1.06987
Xg_16	1.13256
SVD	1.06987
SVD++	1.07029
Xg_18	1.18148

TABLE 2: Comparative study of Models

VI. CONCLUSION

In this project, we have highlighted various techniques and algorithms used in building Recommender Systems, including content-based filtering and collaborative filtering. We have also discussed data pre-processing, feature engineering, and evaluation metrics in building an efficient Recommender System. Furthermore, it provides a detailed comparison of the different techniques used and their effectiveness in different scenarios. The goal of this project was to study the approach used by the winner of the Netflix prize challenge and we have been successful in doing so.

REFERENCES

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," IEEE Computer, vol. 42, DOI 10.1109/MC.2009.263, no. 8, pp. 30–37, 2009.

[2] H. Wang, "Dotmat: Solving cold-start problem and alleviating sparsity problem for recommender systems," 2022.

[3] [Online]. Available: https://surpriselib.com

[4] "https://en.wikipedia.org/wiki/sparse."

[5] "https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr$_matrix.html.''$

[6] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," Commun. ACM, vol. 35, DOI 10.1145/138859.138867, no. 12, p. 61–70, Dec. 1992. [Online]. Available: https://doi.org/10.1145/138859.138867

[7] "https://medium.com/dataseries/similarity-and-distance-metrics-for-data-science-and-machine-learning-e5121b3956f8."

[8] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," vol. 4, DOI 10.1145/1644873.1644874, no. 1, Jan. 2010. [Online]. Available: https://doi.org/10.1145/1644873.1644874

[9] [Online]. Available: https://xgboost.readthedocs.io/en/stable/

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," CoRR, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[11] [Online]. Available: https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data

[12] [Online]. Available: https://sifter.org/~simon/journal/20061211.html