NATIONAL ECONOMICS UNIVERSITY
Faculty of Mathematical Economics

REPORT ON RECOMMENDATION SYSTEM

Instructor: Mrs. Nguyễn Thị Quỳnh Giang

Members:

Nguyễn Minh Anh (35%)

Nguyễn Thị Phương (40%)

Lưu Xuân Nghĩa (25%)

Table of contents

Abbreviation Table

Abbreviation	Meaning
CF	Collaborative Filtering
CBF	Content-Based-Filtering
SVD	Singular Value Decomposition

List of Tables

I) INTRODUCTION

1.1) Background

The explosive growth in the amount of available digital information and the number of visitors to the Internet have created a potential challenge of information overload which hinders timely access to items of interest on the Internet. Information retrieval systems, such as Google, Devil Finder and Altavista have partially solved this problem but prioritization and personalization (where a system maps available content to user's interests and preferences) of information were absent. This has increased the demand for recommender systems more than ever before. Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragments out of a large amount of dynamically generated information according to user's preferences, interest, or observed behavior about items. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile. Recommender systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. Recommendation systems have also proved to improve decision making process and quality. In e-commerce settings, recommender systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

1.2) Significance of Study

Recommendation system has played an imperative role to the success of many companies working in the fields in which the application of the recommendation system is vital. Recommendation system application therefore has strong and direct impacts on not only business targets, but also business strategies and the company's improvement in the feature. Therefore, the final results in this report can be used as a reference for individuals or institutions concerned about how effective a recommendation system can be to their business and can draw a general picture about the recommendation system.

1.3) Research Objectives

The project's major objective is to utilize the support of machines learning models and also traditional models of recommendation system to forecast rating prediction using MovieLens1M dataset from GROUPLEN on Kaggle (Dataset), which is a group of researchers in the Department of Computer Science and Engineering at the University of Minnesota. The other objective is to follow logical analytical procedures: including data collection, data-splitting, model selection, model evaluation… and make sound conclusions based on the model's results.

II) LITERATURE REVIEW

2.1) Recommender Systems

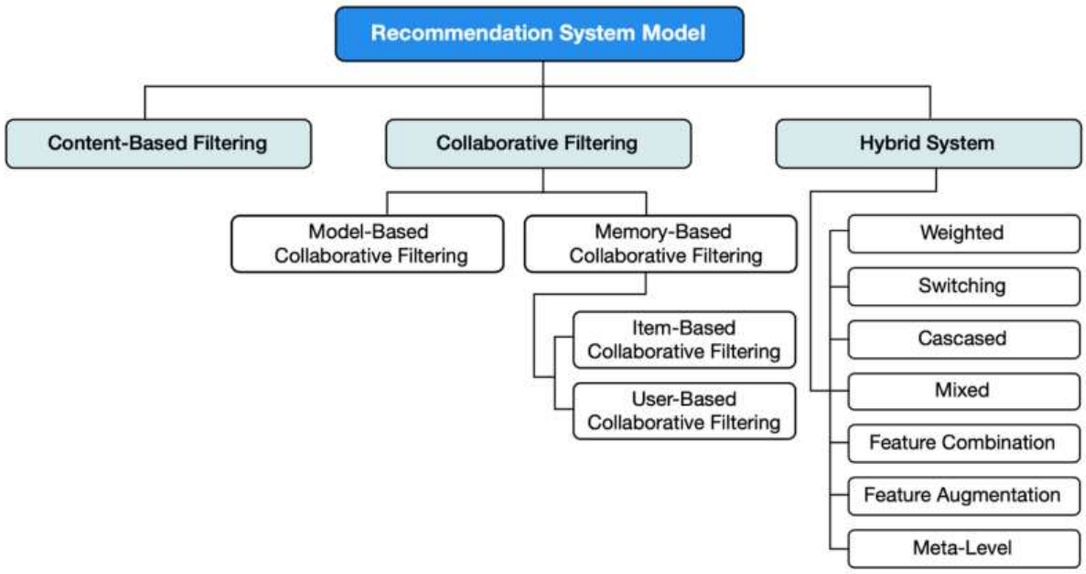
A recommender system is a way for users to make decisions in complicated information situations. It was also characterized from the standpoint of E-commerce as a tool that assists consumers in searching through k nowledge data relevant to their interests and preferences. The recommender system was characterized as a way of supporting and supplementing the social process of making decisions based on the suggestions of others when personal knowledge or experience of the alternatives is insufficient. Recommender systems help consumers deal with the problem of information overload by providing them with individualized, unique content and service suggestions. Various methods for developing recommendation systems have recently been created, including collaborative filtering, content-based filtering, and hybrid filtering.

The collaborative filtering approach is the most developed and widely used. It suggests things by detecting other users who have similar tastes to the current user and using their feedback to make recommendations. Collaborative recommender systems have been used in a variety of settings. Amazon, for example, improves its suggestion by using subject diversity algorithms. To solve the scalability problem, the system employs collaborative filtering, which generates a table of comparable objects offline using an item-to-item matrix. The technology then suggests more things that are comparable online based on the user's previous purchases.

Content-based approaches, on the other hand, match content resources to user attributes. Content-based filtering approaches usually make predictions based on the information provided by the user, and generally disregard contributions from other users, unlike collaborative strategies. In order to construct a training set, Fab primarily relies on user ratings, and it is an example of a content-based recommender system. Letizia is another system that uses content-based filtering to assist people in finding information on the Internet. The system uses a user interface to aid people in exploring the Internet; it can track a person's browsing habit and forecast which pages they would be interested in.

Despite the effectiveness of these two filtering procedures, they have numerous drawbacks. Limited content analysis, overspecialization, and data sparsity are some of the issues associated with both collaborative methodologies and content-based filtering strategies... The quality of suggestions is frequently lowered as a result of these issues. Hybrid filtering, which combines two or more filtering approaches in different ways to boost the accuracy and performance of recommender systems, has been proposed to minimize some of the shortcomings observed. These methods integrate two or more filtering processes in order to maximize their benefits while minimizing their drawbacks. Weighted hybrid, mixed hybrid, switching hybrid, feature-combination hybrid, cascade hybrid, feature-augmented hybrid, and meta-level hybrid are the classifications based on their operations. Ghazantar and Pragel-Benett present a hybrid recommendation approach that leverages an individual user's content-based profile to discover comparable users with whom to make predictions.

Here is the picture which demonstrates for recommendation system approaches:



2.1.1) Collaborative filtering

Collaborative filtering is a domain-agnostic prediction approach for material that cannot be effectively characterized by metadata, such as movies and music. Collaborative filtering saves user preferences for entries in a database (user-item matrix). It then produces suggestions based on profile similarities, which match individuals with appropriate hobbies and preferences. A neighborhood is formed by users who fit into this category. A user may receive suggestions for things that he has not yet evaluated but which have earned positive feedback from other users in his region. Collaborative filtering's advice might take the form of a forecast or a suggestion. Recommendation is a list of the top N things that the user is most likely to enjoy, whereas prediction is a numerical number, Rij that reflects the expected score of item j for user i. There are two sorts of collaborative filtering techniques: memory-based and model-based. based.

2.1.1.1) Memory-based-technique

The items that the user has previously rated are important in his search for a neighbor who appreciates the same things that he does. Once a user's neighbor has been identified, a variety of algorithms can be used to combine neighbors' preferences to generate recommendations. These techniques have found widespread success in real-world applications due to their effectiveness. User-based or item-based techniques can be used to perform memory-based Collaborative filtering.

2.1.1.1.a) User-based nearest neighbor recommendation

The items that the user has previously rated are important in his search for a neighbor who appreciates the same things that he does. Once a user's neighbor has been identified, a variety of algorithms can be used to combine neighbors' preferences to generate recommendations. These techniques have found widespread success in real-world applications due to their effectiveness. User-based nearest neighbor recommendation is

the first approach and one of the earliest methods used in recommendation systems. There has been many researches, papers on this technique such as: A Study on Movie Recommendations using Collaborative Filtering-2021 (Rahul Pradhan, Ashish Chandra Swami, Akash Saxena and Vikram Rajpoot), A Review Paper On Collaborative Filtering Based Moive Recommedation System-2019 (Nirav Raval, Vijayshri Khedkar)… Given a ratings database and the current (active) user's ID as input, the main concept is to locate other users (also known as peer users or nearest neighbors) who had similar preferences to the active user in the past. The forecast is then derived for each product p that the active user has not yet viewed, based on peer user ratings. Such strategies assume that people who have similar tastes in the past will have similar tastes in the future, and that user preferences are stable and constant across time.

There are several methods for computing user similarity with the goal of identifying other users who share the same interests as the target users, including the Pearson coefficient, Cosine similarity…, and the latter, which is used in this report. Since it was proven that cosine similarity produces the most accurate results, it has become the industry standard. Based on their angle, the metric calculates the similarity of two n-dimensional vectors. This metric is also commonly used to compare two text documents that are represented as vectors of terms in information retrieval and text mining.

The similarity between two users *a* and *b*, represented by their respective rating vectors a and b, is formally defined as follows:

$$\text{sim}(a, b) = \cos(a,b) = \frac{\vec{a}.\vec{b}}{|\vec{a}| * |\vec{b}|}$$

The "·" symbol is the dot product of vectors. $|\vec{a}|$ is the Euclidian length of the vector, which is defined as the square root of the dot product of the vector with itself.

After the similarities among users, rating predictions are made by the following formula:

$$\hat{r}_{i,u} = \frac{\Sigma_{u_j \in N(u,i)} r_{i,u_j} * sim(u, u_j)}{\Sigma_{u_j \in N(u,i)} . |sim(u, u_j)|}$$

N(u,i) is the set of k users who have the highest similarity with target user and had rated the item i. For example, there is a ratings database for collaborative recommendation with the Alice's rating for Amelie movie is unknown. The goal here is to predict the rating score that Alice would give to Amelie.

	Matrix	*Alien*	*Serenity*	*Cablanca*	*Amelie*
Alice	5	3	4	4	?
*User*1	3	1	2	3	3
*User*2	4	3	4	3	5
*User*3	3	3	1	5	4
*User*4	1	5	5	2	1

From this data, replacing the unknown Alice's rating for Amelie with zero and computing cosine similarity among users. For instance the similarity between Alice and User1 is calculated as follows:

$$\text{sim(Alice, User1)} = \frac{5*3+3*1+4*1+4*3+0*3}{\sqrt{5^2+3^2+4^2+4^2+0^2}\sqrt{3^2+1^2+1^2+3^2+3^2}} = 0.777156$$

With the help of computer, computing the other similarities gets easier. The below matrix is user-user similarity matrix with entries are similarity between users and users.

	$Alice$	$User1$	$User2$	$User3$	$User4$
$Alice$	1	0.77	0.81	0.76	0.78
$User1$	0.77	1	0.92	0.95	0.54
$User2$	0.81	0.92	1	0.89	0.77
$User3$	0.76	0.95	0.89	1	0.63
$User4$	0.79	0.54	0.77	0.63	1

As shown in the matrix, User2 and User 4 have the highest similarity with Alice then using the rating information of User2 and User4 to forecast how much score will Alice give to Amelie film.

$$\hat{r}_{\text{Amelie,Alice}} = \frac{r_{\text{Amelie,User2}} * sim(Alice, \text{User}_2) + r_{\text{Amelie,User4}} * sim(Alice, \text{User}_4)}{|sim(Alice, \text{User2})| + |sim(Alice, \text{User4})|}$$

$$\hat{r}_{\text{Amelie,Alice}} = \frac{5*0.81+1*0.78}{|0.81|+|0.78|} = 3.025$$

Hence Alice rating for movie Amelie can be approximately 3.

2.1.1.1.b) Item-based nearest neighbor recommendation

Although user-based collaborative filtering approaches have been successful in a variety of domains, there are still some significant challenges when it comes to large e-commerce sites with millions of users and millions of catalog items to manage. The need to scan a large number of potential neighbors, in particular, makes real-time prediction impossible. Large-scale e-commerce sites, on the other hand, frequently use item-based recommendation, which is better suited for offline preprocessing and thus allows for real-time recommendation computation even for very large rating matrices (Sarwaret al. 2001).

The main idea behind item-based algorithms is to compute predictions based on item similarity rather than user similarity. In this report, item-based recommendation approaches use cosine similarity to find similar items, and then use the result to compute rating predictions using the formula:

Cosine similarity: $\text{sim}(a, b) = \cos(a, b) = \frac{\vec{a}.\vec{b}}{|\vec{a}|*|\vec{b}|}$

Rating prediction: $\hat{r}_{i,u} = \frac{\Sigma_{i \in N(u,i)} r_{i_j,u} * \text{sim}(i,i_j)}{\Sigma_{i_j \in N(u,i)} . |\text{sim}(i,i_j)|}$

a and b are the corresponding rating vectors, while N(u,i) is the set of k items with the highest similarity to the target item and which have been rated by a specific user u.

In this item-based example, the data about users and the user's film rating is used once more. Calculating the item-to-item similarity matrix by replacing the unknown of Alice's rating for the film Amelie with zero. The item-based approach is now being used to compute Alice's prediction for the film Amelie.

	Matrix	*Alien*	*Serenity*	*Cablanca*	*Amelie*
Alice	5	3	4	4	?
*User*1	3	1	2	3	3
*User*2	4	3	4	3	5
*User*3	3	3	1	5	4
*User*4	1	5	5	2	1

For instance, the cosine similarity between Amelie and Matrix is computed as:

$$sim(Amelie, Matrix) = \frac{5*0+3*3+5*4+4*3+1*1}{\sqrt{0^2+3^2+5^2+4^2+1^2}\sqrt{5^2+3^2+4^2+3^2+1^2}} = 0.76$$

Using computer the item-to-item matrix is generated as follows:

	Matrix	*Alien*	*Serenity*	*Cablanca*	*Amelie*
Matrix	1	0.78	0.79	0.94	0.76
Alien	0.78	1	0.94	0.84	0.67
Serenity	0.79	0.94	1	0.75	0.58
Cablanca	0.94	0.84	0.75	1	0.81
Amelie	0.76	0.67	0.58	0.81	1

The item-to-item matrix has shown that *Cablanca* and Matrix are top 2 highest similarity with Amelie, so these will be used to produce the rating prediction of Alice for Amelie:

$$\hat{r}_{\text{Amelie,Alice}} = \frac{r_{\text{Matrix,Alice}} * \text{sim}(\text{Matrix}, \text{Amelie}) + r_{\text{Cablanca,Alice}} * \text{sim}(\text{Cablanca}, \text{Amelie})}{|\text{sim}(\text{Mtrix}, \text{Amelie})| + |\text{sim}(\text{Cablanca}, \text{Amelie})|}$$

$$\hat{r}_{\text{Amelie,Alice}} = \frac{5*0.76+4*0.81}{|0.81|+|0.76|} = 4.48$$

Hence, the rating prediction of Alice for Amelie used the item-based approach is almost 5 Amazon.com, an e-commerce recommendation engine that uses scalable item-to-item collaborative filtering techniques to recommend online products for different users, is a shining example of collaborative filtering technique. The computational algorithm scales

independently of the database's size and number of users and items. To obtain information from users, Amazon.com employs an explicit information collection technique. The following sections make up the interface: your browsing history, rating these items, and improving your recommendations and profile. Based on the items he or she has rated, the system predicts the user's interest. The system then compares the user's browsing patterns on the system and recommends an item of interest to the user. The feature of " people who bought this item also bought these items" was also popularized by Amazon.com.

2.1.1.1.c) Neighborhood selection problem

Not all neighbors are considered in the two examples (neighborhood selection). The example intuitively included only those that had a high similarity with the active user/item when calculating the predictions (who had rated the item or which had been rated for which the example was looking for a prediction). If all users/items are included in the neighborhood, not only will the performance suffer in terms of the required calculation time, but the accuracy of the recommendation will suffer as well, as the ratings of other users who aren't really comparable will be taken into account.

The most common methods for reducing the size of the neighborhood are defining a specific minimum threshold of user similarity or limiting the size to a fixed number and only considering the k closest neighbors. Anand and Mobasher (2005) and Herlocker et al. (1999) discuss the potential drawbacks of both techniques: if the similarity threshold is set too high, the size of the neighborhood for many users will be very small, implying that no predictions can be made for many items (reduced coverage). When the threshold is set too low, however, the size of the neighborhood does not shrink significantly.

Coverage is unaffected by the value chosen for k, which is the size of the neighborhood. However, there is still a problem with finding a good value for k: when the number of neighbor k is too large, too many neighbors with low similarity introduce additional "noise" into the predictions. When k is too small – for example, below 10 in the Herlocker et al. (1999) experiments – the quality of the predictions may suffer. According to the MovieLens dataset, "a neighborhood of 20 to 50 neighbors seems reasonable in most real-world situations" (Herlocker et al.2002).

As a result, deciding on k- the size of the neighborhood is another difficult task that developers must carefully consider in order to save time, effort, and improve the accuracy of predictions.

2.1.1.1.d) Neighborhood-based method and normalization

Many factors, such as regression-based vs. classification-based rating prediction, user-based vs item-based approach, etc.., must be considered when implementing neighborhood-based recommendations, since these decisions impact the overall quality of the recommendations. Other key aspects, such as rating normalization, computation of similarity values, neighborhood selection, and so on, can have a substantial influence on the recommendation process.

This study examines the most typical variants for each of these components, as well as their benefits and drawbacks.There are two prevalent methods for rating normalization that take into account an individual's differences in measuring the same amount of appreciation via varied ratings: mean-centering and z-score.

In mean-centering method, the raw ratings are turned into positive or negative ratings, which express the user's admiration for an item directly through the sign of the normalized rating. Raw rating r_{u_i} is transformed to mean-centered rating $h(r_{u_i})$ in a user-based technique by subtracting average user rating $\bar{r_u}$ from it as follows:

$$h(r_{u_i}) = r_{u_i} - \bar{r_u}$$

Therefore, the user-based prediction is obtained by the equation below:

$$\hat{r}_{i,u} = \bar{r_u} + \frac{\Sigma_{u_j \in N(u,i)} (r_{i,u_j} - \overline{r_{u_j}}) * sim(u, u_j)}{\Sigma_{u_j \in N(u,i)} . |sim(u, u_j)|}$$

The same process applied for item-based approach and the item-based prediction is:

$$\hat{r}_{i,u} = \bar{r_i} + \frac{\Sigma_{i \in N(u,i)(} r_{i_j,u} - \bar{r_{i_j}}) * sim(i, i_j)}{\Sigma_{i_j \in N(u,i)} . |sim(i, i_j)|}$$

Z-score normalization is different from the mean-centered since the scheme considers the spread of the ratings and is defined as follows in the user-based approach:

$$h(r_{u_i}) = \frac{(r_{u_i} - \bar{r_u})}{\sigma_u}$$

Hence, the rating prediction is computed using :

$$\hat{r}_{i,u} = \bar{r_u} + \sigma_u \frac{\Sigma_{u_j \in N(u,i)} \frac{(r_{i,u_j} - \overline{r_{u_j}})}{\sigma_{u_j}} * sim(u, u_j)}{\Sigma_{u_j \in N(u,i)} . |sim(u, u_j)|}$$

The rating forecast used in item-based therefore can be defined as:

$$\hat{r}_{i,u} = \bar{r_i} + \sigma_{i_j} \frac{\Sigma_{i \in N(u,i)} \frac{r_{i_j,u} - \bar{r_{i_j}})}{\sigma_{i_j}} * sim(i, i_j)}{\Sigma_{i_j \in N(u,i)} . |sim(i, i_j)|}$$

Since in the utility matrix where each row represents for item and each column illustrates for each user, then subtracting the average from each column causes positive and negative values in each column. Positive values correspond to the users who like the item, negative values demonstrates for those who to do not have interest in the item and values of 0 corresponds to undetermined whether the user likes the item or not. Technically, the number of dimensions of the utility matrix is huge with millions of users and items, saving all these values in a matrix will lead to a high probability that there will not be enough memory. As it is observed that since the number of known ratings is usually a very small number compared to the size of the utility matrix, it is better and wise choice when saving this matrix as sparse matrix, that means saving only other values and their location. Therefore, it will be much optimal to replace the unknown ratings with a '0' value, which is demonstrates for whether the user has interest in the item or not. This technique will not only optimize memory, but also make the calculation of similarity matrix later more effective.

2.1.1.2) Model-based-technique

2.1.1.2.a) SVD - Singular Value Decomposition

Singular value decomposition (SVD) is a collaborative filtering method which is widely used in recommender system for: movies, music, retail product… This technique has gained a lot of interest and is utilized as a powerful matrix factorization approach in many papers such as**:** Building a Movie Recommendation System using SVD algorithm (Asoke Nath , Adityam Ghosh, Arion Mitra), Recommender Systems using SVD with Social Network Information (Min-Gun Kim, Kyoung-jae Kim)…

The SVD method is a type of matrix factorization. It is a method for factoring matrices into distinct components that can be applied to any matrices, whether square or rectangular, degenerate, or non- singular. Singular decomposition is a useful and powerful method of matrices processing because of its broad applicability. "*When used in a recommendation system, SVD is a method of internally projecting the user's and the product's rating score matrix to a specific dimension of user and product combining latent factor space*" (**Ricci et al., 2011**)

Consider a user-item matrix named A, which consists of user u and product i and the elements of matrix A represent each user's rating for each product. The matrix A is decomposed into several small matrices using SVD, the formula given matrix A with m x n shape is presented as follows:

$$A = U\Sigma V^T$$

Matrix U is called m x m left singular vectors which contains eigenvectors of AA^T. U represents the relationship between users and latent factor. Matrix V is called n x n right singular vectors that contains eigenvectors of A^TA. V indicates the similarity between items and latent factors. Matrix Σ is m x n diagonal matrix which contains square root of eigenvalues of matrix with entries are identical non-zero eigenvalues and are arranged in descending order. It describes the strength of each latent factor.

The latent factor or concept is a broad idea which describes a property that a user or an item have. For instance, for music, latent factor can refer to the genre that the music belongs to. To indicate, here is A- a target matrix for singular value decomposition. The objective now is finding U, V and Σ to decompose A

$$A=\begin{bmatrix}3 & 2 & 2\\ 2 & 3 & -2\end{bmatrix}$$

Step1: Finding Σ

To find the elements of Σ, the first step is calculating AA^T:

$$AA^T=\begin{bmatrix}17 & 8\\ 8 & 17\end{bmatrix}$$

From this result, calculating the eigenvalues of this matrix by finding the roots of the following characteristic polynomial:

$$\det(AA^T-\lambda I)=0 \quad <=> \quad \lambda^2-34\lambda+225=(\lambda-25)(\lambda-9)=0$$

Since Σ in SVD is the diagonal matrix that contains the square roots of the eigenvalues of AA^T, hence:

$$\sigma_1=5, \quad \sigma_2=3$$

Where σ_n denotes the value of the nth diagonal entry in Σ. Therefore, given the dimensionality of $A(2x3)$:

$$\Sigma=\begin{bmatrix}5 & 0 & 0\\ 0 & 3 & 0\end{bmatrix}$$

Step2: Finding U

From eigenvalues $\lambda_1=25$, $\lambda_2=9$ have found, substituting respectively λ into the equation $(AA^T: -\lambda I)\ \mathbf{x} = \mathbf{0}$ and solve for **x**, the resulting nonzero solutions form the set of eigenvectors of AA^T corresponding to the selectd eigenvalue hence it's also the column vector of U.

First, given $\lambda_1=25$ then:

$$AA^T-\lambda_1 I=\begin{bmatrix}-8 & 8\\ 8 & -8\end{bmatrix}$$

Computing the result of equation: $(AA^T - \lambda_1 I)\ \mathbf{x} = 0$, the first column vector of U is $\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$

Doing the same for $\lambda_2=9$, the second column vector of U is $\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}$

Hence: $$U=\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}$$

Step 3: Finding V:

Repeating the procedure for $A^T A$ to obtain the factor V, we get matrix V:

$$V=\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{18}} & \frac{2}{3} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{18}} & \frac{-2}{3} \\ 0 & \frac{4}{\sqrt{18}} & \frac{-1}{3} \end{bmatrix}$$

Finally, the original target matrix A is able to be decomposed by 3 matrices: U, V, Σ

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{18}} & \frac{-1}{\sqrt{18}} & \frac{4}{\sqrt{18}} \\ \frac{2}{3} & \frac{-2}{3} & \frac{-1}{3} \end{bmatrix}$$

The original matrix can be approximated by smaller dimension matrices when reducing the dimension of the matrix U,V, Σ. The dimensionally reduced matrix is constructed by extracting only the important features of the original matrix. Singular value decomposition's actual beauty resides in its capacity to compress data by extracting useful information from the input data.

The key to dimensionality reduction in the example above is that when the diagonal entries of the eigenvalues get lower, the first few columns of U, its associated eigenvalues in, and the corresponding first few rows of V^T contain the most information about matrix A. According to the rule of thumb, the smaller the eigenvalue, the less it helps to conveying data about A. In other words, it is completely capable of producing an estimate of A by extracting the first few columns and rows of each element. Consider the following scenario:

$$A \approx \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} [5] \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

It becomes feasible to deconstruct into low-dimensional matrices by removing simple and non-critical elements. In other words, during the recommendation process, part of the noise in the assessment data may be eliminated, and the pure characteristic factor values can be retrieved, lowering the data's complexity.

In general, SVD has the advantage of removing noise and extracting the singular value to express the factor characteristic that is difficult to express in the original matrix, but it also has the disadvantage of losing some information in the process. Furthermore, it has the disadvantage of not utilizing other implicit data by relying solely on the user's product rating data, which is explicitly existing explicit data.

2.1.1.2.b) SVDpp

In personalized recommendation systems, collaborative filtering (CF) has proven to be effective. To improve the accuracy of prediction by generating implicit feedback, the singular value decomposition (SVD)++ algorithm is used as an optimized SVD algorithm. The SVD++ algorithm, on the other hand, is constrained by the recommendation's low calculation efficiency. This study proposes a novel method to accelerate the computation of the SVD++ algorithm, which can help achieve more accurate recommendation results, in order to address this limitation of the algorithm. The proposed method entails using the SVD++ algorithm to perform a backtracking line search, optimizing the recommendation algorithm, and finding the best solution using the backtracking line search on the local gradient of the objective function. In the FilmTrust, MovieLens 1M, and 10M public datasets, the algorithm is compared to the traditional CF algorithm. By comparing the root mean square error, absolute mean error, and recall rate simulation results, the effectiveness of the proposed method is demonstrated. An optimized algorithm based on SVD is referred to as SVD++. SVD is a common matrix decomposition technique that extracts algebraic features effectively. In CF, SVD is used to determine the scorer's preference for each factor as well as the extent to which the film contains each factor in the current scoring scenario. Finally, the data is analyzed one by one to arrive at the predicted outcome.

When solving the minimum value in the SVD++ algorithm, the fixed step size is commonly used. Though it can efficiently reach the minimum value under the extremely large value, it cannot reach the optimal solution based on a few iterations, or even Convergence, so the optimal solution cannot be obtained. Though the optimal solution can be found if the value is too small, there are many iterations and the rate of decline is too slow, resulting in a long time and low efficiency.

As a result, this research proposes using a backtracking line search. Backtracking line search is a method of determining the maximum amount of movement possible in a given descent direction. Starting with a guess when the step value in the descending direction is relatively large, the step size is iteratively reduced until the objective function is identified as being sufficiently reduced based on the gradient. The following conclusions must be reached:

$$f(x - \gamma \nabla f(x)) > f(x) - \alpha\gamma \parallel \nabla f(x) \parallel_2^2$$

$$\gamma = \beta * \gamma$$

$$0 < \beta \leq 1, 0 < \alpha \leq \frac{1}{2}x$$

The step size is updated after the backtracking condition is satisfied, so the objective function is constantly updated in the falling direction. The gradient descent introduces backtracking in the SVD++ algorithm, so this study proposes an optimized backtracking algorithm. If the backtracking condition is not met, the following result is obtained:

$$\gamma = \eta * \gamma, \ 0<\eta\leq 1$$

The step size gradually shrinks as the number of iterations increases, and the results become more accurate. If only the backtracking condition is satisfied, the loop will jump out without satisfying the backtracking condition, reducing the accuracy of the result significantly. Because the backtracking line search's initial value is large, the descent speed is fast at first, and the backtracking line search can easily jump out of the loop. Then, because the backtracking condition is not satisfied, the objective function's minimum value is not the best solution. When the backtracking condition is not met, this formula reduces the step size until the backtracking condition is met or the number of iterations is reached. As a result, the iterative result is more accurate than the random gradient drop because the initial step size is larger. As a result, the initial time is more efficient than the random gradient drop. The backtracking line search is solved in the later part of the iteration by adding the Equation because the gradient descent is not satisfied. The backtracking line search exits the loop, allowing for a smaller step size to yield a more precise optimal solution.

The most likely outcome is for the objective function's value to decrease. The objective function may jump out of the loop directly if the search step is too large. If the search step size is too small, the objective function may iterate on a value indefinitely, making finding the best solution difficult.

In the added function $f(x_k - \gamma d_k)$, on a certain x, after the search direction d_k is determined, then, it only needs to find a suitable step size. Therefore, this is a nonlinear function, with γ as its independent variable. If equation is satisfied, then the following formula will be satisfied:

$$f(x_k - \gamma_k d_k)\big) \leq f(x_k) - \alpha_1 \gamma_k \nabla f(x_k)^T d_k$$

$$f(x_k - \gamma_k d_k)\big) \leq f(x_k) - \alpha_2 \gamma_k \nabla f(x_k)^T d_k$$

Where the inequality is Taylor expansion on the left, it yields:

$$f(x_k - \gamma_k d_k) = f(x_k) - \gamma_k g_k^T d_k + o(\gamma_k)$$

Since $g_k^T d_k < 0, f(x_k) - \lambda_k \rho g_k^T d_k < f(x_k)$ then this objective function will inevitably decrease, and the learning rate will be adaptively changed. First, a larger value step size is used in the gradient descent so that the objective function can be reduced faster during the initial iteration and quickly dipped to the optimal solution. The learning rate gradually decreases as the number of iterations increases, and the optimal solution can be found with a low learning rate, allowing the optimal solution to be found in fewer iterations.

In SVD, decomposing an m∗n matrix R_{mn} needs to be decomposed into an m*m matrix P_{mn} and an m∗n matrix Q_{mn}. The two decomposition matrices do not reduce the dimensions of the original matrix much. The multiplication and iteration of the two matrices result in a time complexity of SVD of $O(mn^2 + nm^{2)}$. According to the definition of time complexity, the time of SVD can be obtained and the complexity is O(n^3).

It is known that RSVD and SVD++ algorithms are obtained by adding regularization and implicit vectors through the SVD algorithm. Even if the implicit feedback of users and items is added, the prediction score matrix acquisition does not need to rely too much on matrix decomposition matrix P_{mn} and matrix Q_{mn}, but the acquisition of implicit vectors still requires *m* times of user implicit vector acquisition and acquisition in matrix R_{mn} *n* times item implicit vector acquisition. Therefore, RSVD and SVD++ have the same time complexity as SVD, which is O(n^3).The BLS-SVD++ algorithm proposed in this paper adds backtracking judgment in the loop process, and does not add the loop and the time complexity is $O(n^3 + b)$, that is, in the worst case, the time complexity is O(n^3). That is the same time complexity as the SVD++ algorithm.

Equation is added to make the objective function search for more optimal advantages in the gradient descent process after ensuring that the loss function decreases. The decline rate is fast at first because the initial value of the step size is large, and it is easy to jump out of the loop in a short time, allowing the final non-optimal solution to be solved. Thus, Equation is added to reduce the step size and ensure a more accurate final result.

2.1.2.3) Pros and Cons of collaborative filtering techniques

Collaborative Filtering provides a number of advantages over Material-based filtering, including the ability to function in domains with little content connected with things and content that is difficult for a computer system to assess (such as opinions and ideal). In addition, the collaborative filtering approach may deliver serendipitous suggestions, which means it can suggest items that are interesting to the user even if the material is not in the user's profile. Despite the efficacy of collaborative filtering methods, their broad usage has exposed a number of possible issues, including the ones listed below.

a. Cold-start problem

This is a circumstance in which a recommender lacks sufficient knowledge about a user or an item to make accurate predictions. This is one of the primary issues that causes the recommendation system's performance to suffer. The profile of such a new person or item will be empty because he has not yet rated any items, hence the system is unaware of his preferences.

b. Data sparsity problem

This is a problem that arises when there is insufficient information, when only a small percentage of the total number of entries in a database are evaluated by users. This invariably results in a sparse user-item matrix, the inability to find successful neighbors, and the development of poor suggestions. Furthermore, data scarcity always causes coverage issues, which refers to the percentage of objects in the system for which suggestions may be given.

c. Scalability

Another issue with recommendation algorithms is that computation expands linearly in proportion to the number of users and items. When the number of datasets is restricted, a recommendation approach may be unable to create a sufficient number of suggestions when the dataset capacity is raised. As a result, it's critical to use recommendation systems that can scale up successfully as the quantity of datasets in a database grows. Methods for tackling the scalability problem and speeding up suggestion creation are based on dimensionality reduction techniques like the Singular Value Decomposition (SVD) method, which can yield trustworthy and fast results and efficient recommendations.

d. Synonymy

The tendency for highly similar goods to have distinct names or entries is known as synonymy. The distinction between closely similar goods, such as the difference between baby clothing and baby cloth, is challenging for most recommender systems. To compute their similarity, collaborative filtering methods frequently discover no match between the two phrases. The synonymy problem may be solved using a variety of ways, including automated word expansion, the creation of a thesaurus, and Singular Value Decomposition (SVD), particularly Latent Semantic Indexing. The disadvantage of these strategies is that some new phrases may have different meanings than intended, resulting in quick loss in recommendation performance.

2.1.2) Content-based filtering

2.1.2.1) Approach to content-based filtering

The content-based technique is a domain-specific algorithm that focuses on analyzing item attributes in order to generate predictions. When it comes to recommending documents like web pages, publications, and news, the content-based filtering technique is the most effective. The content-based filtering technique makes recommendations based on user profiles and features extracted from the content of items the user has previously evaluated. The user is recommended items that are mostly related to the positively rated items. In order to generate meaningful recommendations, CBF employs a variety of models to find similarities between documents. Content-based filtering can be approached in three ways:

- Content presentation and content similarity (VSM and TF-IDF)
- Text classification and methods (Probabilistics method)
- Similarity-based retrieval

To model the relationship between different documents within a corpus, it could use Content presentation and content similarity approaches with Vector Space Models such as Term Frequency Inverse Document Frequency (TF/IDF) or Text classification and methods with Probabilistic models such as Naive Bayes Classifier Decision Trees or Neural Networks, or taking advantage of similarity-based retrieval approach with k-Nearest neighbors method to estimate to what extent a certain document is similar to another document. These methods generate recommendations by learning the underlying model through statistical analysis or machine learning. Because other users' profiles do not influence recommendation, the content-based filtering technique does not require them. In addition, if the user profile changes, the CBF technique can still adjust its recommendations in a short period of time. The main disadvantage of this method is that it necessitates a thorough understanding and description of the characteristics of the items in the profile.

In reality, many companies use the benefits of content-based techniques to apply for their recommendation systems in the real world, so content-based techniques are widely used across all industries. IBRA, for example, is a content-based book recommendation system that gathers information about books from the Internet. It uses information extracted from the web to train a Nave Bayes classifier to learn a user profile and produce a ranked list of titles based on training examples provided by a single user. The system is capable of providing explanations for any recommendations made to users by listing the features that contribute to the highest ratings, allowing users to have complete confidence in the system's recommendations.

2.1.2.2) Pros and Cons of content-based filtering technique

The challenges of CF are overcome using CB filtering techniques. Even if no ratings are provided by users, they have the ability to recommend new items. As a result, even if the database does not contain user preferences, the accuracy of recommendations is unaffected. It also has the ability to adjust its recommendations in a short period of time if the user's preferences change. They can handle situations in which different users don't share the same items, but only items that are identical in terms of their intrinsic characteristics. Users can receive recommendations without disclosing their personal information, ensuring their privacy. The CBF technique can also explain to users how recommendations are generated.

However, as discussed in the literature, the techniques have a number of flaws. The metadata of items is used in content-based filtering techniques. That is, before they can make recommendations to users, they need a detailed description of the items and a well-organized user profile. This is referred to as a "limited content analysis." As a result, the efficacy of CBF is contingent on the availability of descriptive data. Another serious issue

with the CBF technique is content overspecialization. Users can only receive recommendations for items that are similar to those already defined in their profiles.

2.1.3. Hybrid filtering

Hybrid filtering combines different recommendation techniques to improve system optimization and avoid some of the drawbacks and issues that come with pure recommendation systems. Hybrid techniques are based on the idea that a combination of algorithms will provide more accurate and effective recommendations than a single algorithm because the disadvantages of one algorithm can be overcome by another algorithm. In a combined model, using multiple recommendation techniques can mask the flaws of each individual technique. The following methods can be used to combine approaches: separate algorithm implementation and combining the results, using some content-based filtering in a collaborative approach, using some collaborative filtering in a collaborative approach, creating a unified recommendation system that combines both approaches.

The below table describe approaches using in hybrid filtering technique:

Table 1. The approaches in hybird filtering

Hybrid Method	**Description**
Weighted Hybridization	A method in which the weight is gradually adjusted according to the degree to which user's evaluation of an item coincides with the evaluation predicted by the recommendation system
Switching Hybridization	A Method of changing the recommendation model used depending on the situation
Cascaded Hybridization	After using one of the recommendation systems models to create a candidate set with a similar taste to the user, the method combines the previously used recommendation system model with another model to sort the candidate set I the order of item most suited to the user's taste
Mixed Hybridization	When many recommendations are made at the same time, content-based filtering can recommend items based on the description of the items without evaluation, but there is a start-up problem in that it cannot recommend new items with insufficient information. To solve this problem, the Mixed Hybridization method recommends items to the user by integrating the user's past history data.

Feature Combination	A collaborative filtering model is used for featured data and example data for items, and a Content-based filtering model is used for augmented data.
Feature Augmentation	A Hybrid method in which one recommendation system is used to classify an item's preference score or item, and the generated information is integrated into the next recommendation system model
Meta-Level	A method of using the entire model of one recommendation system as the input data in the model of another recommendation system. Since the user's taste is compressed and expressed using Meta-Level, it is easier to operate the Collaborative Mechanism than when raw rating data are used as single- input data

2.2). Metrics

The quality of a recommendation algorithm can be evaluated using different types of measurement which can be accuracy or coverage. The type of metrics used depends on the type of filtering technique. Accuracy is the fraction of correct recommendations out of total possible recommendations while coverage measures the fraction of objects in the search space the system is able to provide recommendations for. Metrics for measuring the accuracy of recommendation filtering systems are divided into statistical and decision support accuracy metrics. The suitability of each metric depends on the features of the dataset and the type of tasks that the recommender system will do. In this report, Root Mean Square deviation- RMSE is taken to be the main metric to compare the effectiveness among models. In addition, business metrics are also referred to in this part with the purpose of discovering how well the recommendation works to users and the ability to correctly recommend to users…

2.2.1). Root Mean Square deviation- RMSE

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

$$\text{RMSE}=\sqrt{\frac{\sum(y_i-y_p)^2}{n}}$$

y_i: Actual value.

y_p: Predicted value

n: Number of observations

Statistical accuracy metrics evaluate accuracy of a filtering technique by comparing the predicted ratings directly with the actual user rating. Root Mean Square Error (RMSE) is usually used as statistical accuracy metrics. Root Mean Square Error (RMSE) puts more emphasis on larger absolute error and the lower the RMSE is, the better the recommendation accuracy.

2.2.2) Business metrics

A business metric is a numerical indicator that companies use to track, monitor, and evaluate the performance or failure of various business activities. Business metrics are primarily used to communicate an organization's progress toward certain long- and short-term goals. As a result, it has become a crucial key that is relevant to a company's areas of business. Some organizations outline business metrics in their mission statements.

There are many advantages that a correctly defined business metrics brings to business such as:

- Performance improvement: Tracking the correct business metrics may inform how well or poorly a company is doing and point out the right path for improvement.
- Comparative analysis: Monitoring business metrics tells if the company is outperforming or lagging industry benchmarks.
- Alignment: Business metrics may be used to guarantee that everyone in the business is working toward the same goals.
- Identifying issues: Analyzing company metrics may aid in the early detection of emerging issues, allowing them to be addressed before they become serious issues.

For the purpose of this study, business metrics is defined as the percentage of movies that customers watched based on suggestions from the recommendation system, according to the formula:

$$\text{Business metric} = \frac{\text{total number of rated}-\text{movies that customer watched in the recommendation list}}{\text{total number of movies that customer has rated}} . 100$$

This metric represents actual effectiveness percentage of the recommendation list given to the client. The possible ranges of this metrics is from 0 %- 100%, where

- 0% : the customer has not watched any of the movies in the recommended list
- 100% : customers have watched all the movies in the recommended list

The major advantage of this business metrics is to be able to evaluate the actual efficiency of the algorithm when put into application. Which therefore brings back many benefits for managers since they can depend on the results to make decisions on business operations or answer for the question is recommendation system good enough to boost their business. However, the drawback is that it should not fully be considered as a criterion for judging whether the recommended list is effective or not since the suggested list is given based on the customer's predicted ratings for the movie. But in reality, there are cases where customers have watched the movie but not left a review or rating.

III) METHODOLOGY

3.1. Data Description

The dataset is retrieved from Kaggle. The data contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.There are 3 files included namely: movies.dat, ratings.dat, users.dat which indicates 3 essential information needed for recommendation system approaches. Thanks to Shyong Lam and JonHerlocker for cleaning up and generating the data hence, the dataset, which was loaded from Kaggle, is clean and useful without being processed through the pre-processing data process. Below is a general description for 3 data files: Movie, User, Rating.

MOVIES.DAT: Movie information is in the file "movies.dat" and in the following format: MovieID - Title - Genres

- Titles are name of movie (including year of release)
- Genres are pipe-separated and are selected from the following genres: Action, Adventure, Animation, Children's Comedy…

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

RATINGS.DAT: All ratings are contained in the file "ratings.dat" and are in the following format: UserID - MovieID - Rating - Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch as returned by time
- Each user has at least 20 ratings

	User_ID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275

USERS.DAT: User information is in the file "users.dat" and is in the following format: UserID: Gender - Age - Occupation - Zip-code.

Users supply all demographic information voluntarily, and it is not verified for correctness. This data collection only includes users who have submitted some demographic information.

- Gender is denoted by a "M" for male and "F" for female
- Age is from under 18 to 56+ and is divided into many ranges with specific age labels: 1 is for "Under 18" , 18 is for "18-24", 25 is for "25-34"....
- Occupation is chosen from the many choices and is given a unique label. For instance: 0: "other" or not specified, 1: "academic/educator", 2: "artist"...

	User_ID	Gender	Age	Occupation
0	1	F	1	10
1	2	M	56	16
2	3	M	25	15
3	4	M	45	7
4	5	M	25	20

3.2 Train test split

The data split strategy is divided into 4 options, which includes all possibly potential splitting choices: Random-split-by-ratio, Random-split-by-user, Leave-one-out-split, Split-by-time point. The detail is represented in the below table

Table 2 Data splitting strategy

Data split strategy	**Definition of training and test instances**	**Local timeline**	**Global Timeline**	**Data leakage**
Random-split by-ratio	Randomly sample a percentage of user item interactions as test instances; the remaining are training instances.	No	No	Yes

Random-split by-user	Randomly sample a percentage of users and take all their interactions as test instances; the remaining instances from other users are training instances.	No	No	Yes
Leave-one-out split	Take each user's last interaction as a test instance; all remaining interactions are training instances.	Yes	No	Yes
Split-by-time point	All interactions after a time point are test instances; interactions before this time point are training instances.	No	Yes	No

For the purpose of this report, the splitting strategy: split by time point is chosen since it has proved to be the best fit for the MovieLen1M dataset. This dataset is valid from April 25, 2000 to February 28, 2003, the training set, validation set and test set are divided by percentile. In which, the training set data occupies from per 0 percentile to 70^{th} percentile (from April 25, 2000 to April 23, 2002), the validation set is from 70^{th} percentile to 85^{th} percentile (from April 23, 2002 to September 25, 2002), the test set takes up from 85^{th} percentile to the last one (from September 25, 2002 to February 28, 2003). Following this division, the strategy divides the train set with 70% of the data, 15% for the test set and 15% for the validation set.

3.3) Introduction to Surprise package and model implementation

3.3.1) Overview and advantage of Surprise package

Surprise stands for "Simple Python Recommendation System Engine". Surprise is a Python scikit for developing and analyzing recommender systems based on explicit rating data. It was created with the following goals:

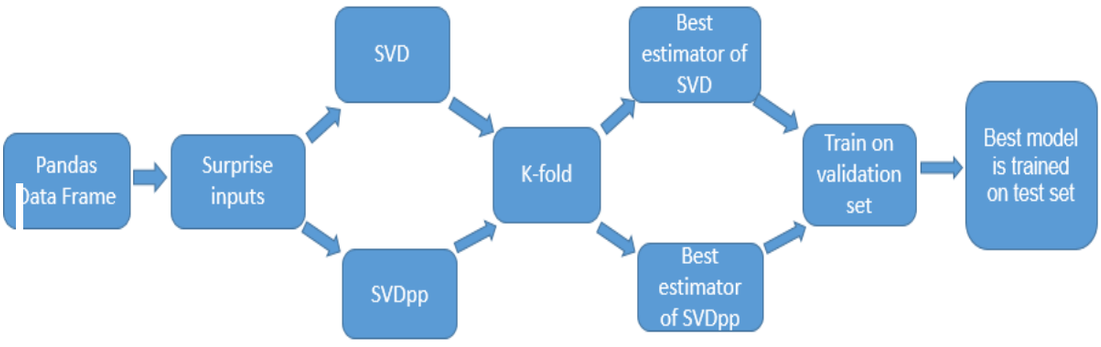
- Allow users complete control over their experiments. To that end, we have placed a strong emphasis on documentation, which we have attempted to make as clear and precise as possible by highlighting every detail of the algorithms.
- Reduce the agony of dealing with Datasets. Users can use both built-in datasets (Movielens and Jester) and custom datasets.
- Prediction algorithms that are ready to use include baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Various similarity measures (cosine, MSD, pearson...) are also included.
- Make it simple to implement new algorithm concepts.

- Provide tools for evaluating, analyzing, and comparing the performance of the algorithms. Cross- validation procedures are very simple to implement, thanks to powerful CV iterators (inspired by scikit-excellent learn's tools) and an exhaustive search over a set of parameters.

3.3.2) Model implementation with Surprise package

In this study, the example of how Surprise package is applied to solve the problems in recommendation system will be implemented along with the 2 models in collaborative filtering technique. Specifically, the model-based knowledge which includes SVD and SVDpp model. The flow is to firstly convert pandas frame into surprise inputs which fits for SVD and SVDpp model. After that, simply implementing SVD and SVDpp model on the inputs and then running the models through K-fold algorithm to get the best estimators parameters of models. The best estimators are then used to train on test set to see how well these models are? Lastly, to compare the two models cross validation on validation dataset is then applied. The metric to compare is RMSE.

Below is the workflow of using Surprise package to implement SVD and SVDpp models:



Before getting down to the work, installing packages, importing necessary libraries, loading datasets are the very first step which need to be done first. Here are pictures demonstrates these simple steps:

- Install scikit-surprise

```
!pip3 install numpy
!pip3 install scikit-surprise
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21.6)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.1.tar.gz (11.8 MB)
     |████████████████████████████████| 11.8 MB 4.7 MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-surprise) (1.1.0)
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.7/dist-packages (from scikit-surprise) (1.21.6)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-surprise) (1.4.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (from scikit-surprise) (1.15.0)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp37-cp37m-linux_x86_64.whl size=1633709 sha256=55355
  Stored in directory: /root/.cache/pip/wheels/76/44/74/b498c42be47b2406bd27994e16c5188e337c657025ab400c1c
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.1
```

- Import library

```
import pandas as pd
import numpy as np
from surprise import Reader
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise import NormalPredictor
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
from surprise import KNNBaseline
from surprise import SVD
from surprise import BaselineOnly
from surprise import SVDpp
from surprise import NMF
from surprise import SlopeOne
from surprise import CoClustering
from surprise.accuracy import rmse
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise.model_selection import GridSearchCV
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
from surprise import Dataset
from surprise.model_selection import KFold
from past.builtins import xrange
from sklearn.metrics import mean_squared_error
```

- Load dataset

```
PATH_TRAIN = '/content/drive/MyDrive/ML/Recommendation_System/Data/ML-1M-SPLIT/TIMESTAMP/train.csv'
PATH_TEST = '/content/drive/MyDrive/ML/Recommendation_System/Data/ML-1M-SPLIT/TIMESTAMP/test.csv'
PATH_VAL = '/content/drive/MyDrive/ML/Recommendation_System/Data/ML-1M-SPLIT/TIMESTAMP/validation.csv'
```

```
train = pd.read_csv(PATH_TRAIN)
test = pd.read_csv(PATH_TEST)
val = pd.read_csv(PATH_VAL)
```

```
train = train[['UserID', 'MovieID', 'Rating']]
test = test[['UserID', 'MovieID', 'Rating']]
val = val[['UserID', 'MovieID', 'Rating']]
```

The first flow is input transformation from pandas data frame to surprise format, in order to build SVD and SVDpp model using surprise package, the inputs must be converted into format which fits the surprise package. In this package, the inputs need to be gone through two main processes.

The first process is using Reader class to parse a file containing ratings. Such a file is assumed to specify only one rating per line, and each line needs to respect the following structure: user; item; rating; [timestamp]. There are several parameters needed to be included in but in this case of study, only the parameter “rating_scale” must be redefined with the scale from 1 to 5 (1,5). Others parameters can be set as default. The command should be Reader(rating_scale=(1, 5)).

The second process is using load_from_df class to load dataset from pandas data frame and format the dataset with Reader which has been done in the first process. Using the following command to comlete the second input process: load_from_df (df, reader). The “df” is the data frame containing the ratings.

Hence, from the original pandas data frame, using built-in class in surprise package, inputs are converted into surprise input format which data frame with all lines having the format of Reader user; item; rating; [timestamp] (value from 1-5). For example, the picture below is demonstrated for transforming trainset into surprise data format.

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train[['UserID', 'MovieID', 'Rating']], reader)
```

In the second flow the goal is applying SVD and SVDpp model with K-fold algorithm to pick the best estimator of model, after transforming data to surprise inputs, SVD and SVDpp model can be applied. The first model to build is SVD. In surprise package, SVD model is built already by developers with command SVD(). Since the target here is to use K-fold algorithm to find the best estimator for SVD model. With the help of many built-in functions in surprise package such as: acurancy.rmse or surprise.model_selections… computing estimator is much easier. SVD model uses the fit() method to train assigned folds, takes advantage of test method to produce rating predictions and then calculates exact RMSE score for each fold. The best estimator is defined as the one that having the lowest RMSE score.

The below picture is used to illustrate for SVD model results after running through 2 folds in K-fold algorithm.

```
def estimator_svd(n_split,inputs):
    from surprise.model_selection import KFold
    from surprise import accuracy
    acc=[]
    kf = KFold(n_splits=n_split)
    algo= SVD()
    models=[]
    for trainset, testset in kf.split(inputs):
        model=algo.fit(trainset)
        models.append(model)
        predictions = algo.test(testset)
        acc.append(accuracy.rmse(predictions, verbose=True))
    return models,acc

modells,accuracy=estimator_svd(2,data)
```

```
RMSE: 0.9041
RMSE: 0.9032
```

The best estimator is found by using the numpy built-in function np.argmin() which returns the index of minimum value in accuracy list which has been used to store the values of RMSE in the previous process.

```
def find_best_estimator_index(arr):
  return np.argmin(arr)
```

```
best_estimator__SVD_index=find_best_estimator_index(accuracy)
```

Hence, the index of best estimator for SVD model is then assigned in the variable best_estimator_SVD_index. The next process is to train SVD's the best estimator on the validation set to see how well the model is?

Before training model on validation set, the validation dataset must be converted into surprise format. This process is a bit different from the transformation from train dataset to surprise inputs. In this step, the validation dataset must be undergone 3 smaller steps. The first is converting the general validation set into surprise inputs by using Reader class and load_from_df function, then using the built-in function build_full_trainset() in Dataset module which is a module in surprise package to convert the previous data to trainset in the surprise format, after this step is done, using build_testset() to create a list of ratings in surprise format which is need for the test method.

All these steps must be done because the test method in surprise package only receives a list of ratings to compute results. Meanwhile, the original test set is not a list of ratings, even extracting the rating columns and transforming it into a list, it is still not in surprise format. Since surprise inputs format loaded from data frame with specific order of user; item; rating; [timestamp], not a list of ratings. Hence, doing these following steps is required to transform test set into surprise format. More correct explanation is to convert the data frame of test set in to surprise format and then extracting the test set's rating to serve for the purpose of using test method.

The below pictures demonstrate for these steps: converting test set into surprise format, using it to train the SVD's best estimator model. In previous steps, SVD models operated through 2 folds and computed RMSE score, then using numpy built-in function to return the index of best estimator which is the second model. Here it (2rd model) is used to train for test set. The result is shown in picture as an example for the result all steps.

```
def create_data_for_val_method(df):
    from surprise import accuracy
    reader1 = Reader(rating_scale=(1, 5))
    data1 = Dataset.load_from_df(df[['UserID', 'MovieID', 'Rating']], reader1)
    TrainSet_Val= data1.build_full_trainset()
    TestSet_Val=TrainSet_Val.build_testset()
    return TestSet_Val

def predict(model,test_set):
    from surprise import accuracy
    pred=model.test(test_set)
    return accuracy.rmse(pred,verbose=True)

Test_set=create_data_for_val_method(val)
predict(modells[best_estimator__SVD_index],Test_set)
```

```
RMSE: 0.9329
0.9328543861731275
```

With the purpose of improve SVD model result, the second model which is better than original SVD named SVDpp is employed to train through the MovieLens1M dataset. The same logic and flow is applied for SVDpp model. The first is convert dataset into surprise format, the second is to find the best estimator for SVDpp in K-Fold algorithm, the last is to train best estimator on the validation to see how well the SVDpp model works.

The below pictures illustrate all steps above with SVDpp model running with 2 folds.

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train[['UserID', 'MovieID', 'Rating']], reader)
```

```
def estimator_svdpp(n_split,inputs):
    from surprise.model_selection import KFold
    from surprise import accuracy
    acc=[]
    kf = KFold(n_splits=n_split)
    algo= SVDpp()
    models=[]
    for trainset, testset in kf.split(inputs):
        model=algo.fit(trainset)
        models.append(model)
        predictions = algo.test(testset)
        acc.append(accuracy.rmse(predictions, verbose=True))
    return models,acc

modells,accuracy=estimator_svdppsvdpp(2,data)
```

```
RMSE: 0.8864
RMSE: 0.8839
```

```
def find_best_estimator_index(arr):
  return np.argmin(arr)
```

```
best_estimator__SVDpp_index=find_best_estimator_index(accuracy)
```

```
def create_data_for_val_method(df):
    from surprise import accuracy
    reader1 = Reader(rating_scale=(1, 5))
    data1 = Dataset.load_from_df(df[['UserID', 'MovieID', 'Rating']], reader1)
    TrainSet_Val= data1.build_full_trainset()
    TestSet_Val=TrainSet_Val.build_testset()
    return TestSet_Val

def predict(model,test_set):
    from surprise import accuracy
    pred=model.test(test_set)
    return accuracy.rmse(pred,verbose=True)

Test_set=create_data_for_val_method(val)
predict(modells[best_estimator__SVDpp_index],Test_set)
```

```
RMSE: 0.9193
0.9192761575686926
```

Finally, the last process in the workflow is to train the best model with the test set. In previous flow, it is found that the model which performs the best on validation set, in this case it is SVDpp, the model then is used to train for the test set to record how well the model works with new unseen data. Using the same flow to train model on test set: converting test set in surprise format the utilizing surprise method build_full_trainset() and build_testset() to extract ratings in test set in order to serve for test method.

The picture below shows 2 function which demonstrate the last flow:

```
def create_data_from_testset(df):
    from surprise import accuracy
    reader1 = Reader(rating_scale=(1, 5))
    data1 = Dataset.load_from_df(df[['UserID', 'MovieID', 'Rating']], reader1)
    TrainSet_Val= data1.build_full_trainset()
    TestSet_Val=TrainSet_Val.build_testset()
    return TestSet_Val

def predict_with_val_data(model,test_set):
    from surprise import accuracy
    pred=model.test(test_set)
    return accuracy.rmse(pred,verbose=True)

Test_set=create_data_from_testset(test)
predict_with_val_data(modells[best_estimator__SVDpp_index],Test_set)
```

```
RMSE: 0.9453
0.9452800209883169
```

IV) RESULTS

This study focuses mainly on collaborative filtering technique with 2 main parts: memory-based with user-based and item-based nearest neighborhood recommendation technique and model-based with SVD and SVDpp models. Each of technique has its own pros and cons. But there is model which performs better than others, this is shown by the RMSE scores which reflects how well the model is. The below table shows the RMSE result:

Table 3 Models result

Algorithms	RMSE (train set)	RMSE (validation set)	RMSE (test set)
Users - Users	0.977	0.955	
Items - Items	1.05	0.972	
SVD	0.932	0.902	
SVDpp	0.919	0.884	0.94545

From the table, 4 models are collaborative filtering models which are classified into 2 main parts, memory-based (2 first models) and model-based (2 last models). For the train set analysis, in memory-based, it is shown that user-based nearest neighborhoods recommendation performed much better than item-based, since it has lower RMSER score than item-based model. "*User-based filtering is expected to be superior when dealing with big amounts of data, whereas item-based collaborative filtering is expected to perform better on smaller datasets",* Comparison of User Based and Item Based

Collaborative Filtering Recommendation Services (2017 Peter Bostrom, Melker). For the dataset used in this study, there is more than 1 million rows of observations, this amount of data can be considered as a large dataset. Hence, user-user model performs better then item-item model is understandable.

In the model-based approach with 2 models SVD and SVDpp, it is said that the SVDpp model has lower RMSE score (0.919) than SVD (0.932). This is because SVDpp is developed to overcome some weaknesses of original SVD. SVDpp is an extension of SVD for implicit ratings. Which means SVDpp focuses more on the rating that users give intuitively not the ratings that users are asked by service provider. These rating are valuable to business since they really reflects behaviors of users.

On the validation set analysis, the same happens when user-user model is better than item-item model (0.955-0.972) and SVDpp exceeds SVD (0.884-0.902). There is a remarkable notice that the RMSE score of all model on validation set is lower than that on training set. Which somehow demonstrates the ability of models when they are trained on new dataset. From which, this study picks the model that has the lowest RMSE score on both training set and validation test to finally train for the test set.

When comparing the 4 models above it can be said that SVDpp ranks the first. This also indicates that SVDpp has better ability in predicting ratings with lower errors than others. Since SVDpp is the best model then it is used to applied test set on.

V) DISCUSSION

For the purpose of the discussion, the RMSE on test set of SVDpp model is larger than that on both train and validation set which may illustrate the overfitting problem. Hence, hyper parameters tuning must be considered in order to improve the model performance.

However, tuning can be very time-consuming since SVDpp model takes many hyper parameters and other materials invloved in tuning process. Since the significance of this study is to mainly provide to individual or institutions common knowledge about recommendation system, to draw a general picture on collaborative filtering approaches and how well the models perform with MovieLens1M dataset. Therefore, if a business wants to apply this SVDpp model to improve the quality of recommendation in the feature, the firm’s data scientists, researchers or managers should consider the trade-offs between time, efforts, cost… and effectiveness or likely applicable ability of this model in reality. It is sure that to prove the effectiveness of a model then it must be experimented through many processes by many data scientist and related professors. Hence, the result of models or any conclusion in this study only can be considered as references for ones who have interest in recommendation system especially, in collaborative filtering technique.

Business metric is another important thing must be discussed in this part since it somehow reflects how good the recommendation are made for users. In this study, the business metric as it has been mentioned in the literature part, is defined as the percentage

of movies that customers watched based on suggestions from the recommendation system, that means business metric is the division within the total number of rated-movie that a customer watched in the recommendation list and the total number of movie that a customer have rated. The business metric result is shown in the table:

Table 4 Business metric result

Business metrics	**100%**	**0%**	**> 50%**	**>80%**
Percentage of customer	6.87%	24.46%	30.04%	9.88%

In the test set with 318 customers, the average business metrics is 32.60%. That means for every 100 movies rated by customers, an average of 33 movies are from the recommended list.

The first column demonstrates that there is 6.87% of customers equivalently to 22 clients in the test set who have rated movies and all these movies are exactly precise with the list of recommendation. Simply this illustrates that out of 100 movies that customers in this group have rated, it can be said that they are recommended to customers by the recommendation system. The second columns is example of 24.46% of customers around 78 clients have not rated any movie that the recommendation system offers. In other word, out of 100 movies that customers have rated, there are not any movies in the list of recommendation.

The percentage in the third column says that 30.04% of customers which is equivalent to 97 people, who have rated movies that matches more than 50% of movies in recommendation list. To put it another way, out of 100 movies rated by customers, over 50 movies are taken from the suggested list. The last 9.88% of customers meaning around 31 users, who have the list of rated movies that coincide with more than 80% movies which exist in recommendation list. Otherwise stated, out of 100 movies rated by customers, there are more than 80 movies coming from the system's suggestion.

It is believed that it totally depends on the manager aspect to determine is this result good or bad. The term "good" or "bad" is relies on how the business context is to each firm. To some of the managers who view this result as references since they have interest in recommendation system can say this is good but other may say not. The result above totally based on a small sample of users in test set hence, it cannot reflect how the recommendation system will perform in the reality.

VI) CONCLUSION

This study focuses on comparing four different collaborative filtering algorithms including: user and item-based nearest neighbors recommendation system, SVD and SVDpp model, in which the aim is to provide individual or institutions with common knowledge about recommendation system, draw a general pictures on collaborative filtering approaches and find out which collaborative filtering model that produced the best rating prediction. The four algorithms has been implemented and mentioned. Out of

the 4 model, each one has its own pros and cons. In this case of study the MoiveLens1M has been used to train these 4 models. This paper also used two 2 metrics to evaluate model: RMSE to determine which model is the best at rating prediction and business metric is used to see how well the model performs following the business aspect. Among four models which can be classified as 2 main parts: memory-based and model-based, with memory-based it is shown that user-based-nearest neighbor recommendation is much better than item-based. For model-based, SVDpp has been proved to exceed SVD model. Out of 4 models, SVDpp is outstanding with the lowest RMSE compared to others However, there are also many issues that should be taken into consideration such as: the overfitting problem of model, the hyper parameter tuning process and correct recommendation improvement.

VII) REFERENCES

1. The link of group code: File
2. A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields, Hyeyoung Ko, Suyeon Lee , Yoonseo Park and Anna Choi.
3. Recommender Systems**,** Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedrich.
4. Neighborhood-based Collaborative Recommendations: An Introduction,__Vijay Vermand Rajesh Kumar Aggarwal.
5. A comprehensive survey of neighborhood-based recommendation methods, Christian Desrosiers and George Karypis.
6. Building a Movie Recommendation System using SVD algorithm Asoke Nath 1, Adityam Ghosh 2, Arion Mitra .
7. Comparison and Improvement Of Collaborative Filtering Algorithms, Victor Hansjons Vegeborn, Hakim Rahmani.
8. Recommendation systems: Principles, methods and evaluation, F.O. Isinkaye Y.O. Folajimi b, B.A. Ojokoh.
9. Konstan JA, Riedl J. Recommender systems: from algorithms to user experience. User Model User-Adapt Interact 2012;22:101–23. Pan C, Li W. Research paper recommendation with topic analysis. In Computer Design and Applications IEEE 2010.
10. Beheshti, A.; Yakhchi, S.; Mousaeirad, S.; Ghafari, S.M.; Goluguri, S.R.; Edrisi, M.A. Towards Cognitive Recommender Systems. *Algorithms* **2020**, *13*, 176. [CrossRef] Abbasi-Moud.
11. Vahdat-Nejad, H.; Sadri, J. Tourism Recommendation System Based on Semantic Clustering and Sentiment Analysis.
12. *Expert Syst. Appl.* **2021**, *167*, 114324. [CrossRef] Liu, F.; Lee, H.J.
13. Use of Social Network Information to Enhance Collaborative Filtering Performance. *Expert Syst. Appl.* **2010**, *37*,4772–4778. [CrossRef] Yang, B.; Lei, Y.; Liu, J.; Li, W.
14. Social Collaborative Filtering by Trust. *IEEE Trans. Pattern Anal. Mach. Intell.*

2017, *39*, 1633–1647.[CrossRef] [PubMed]Amato, F.; Moscato, V.; Picariello,

15. A Multimedia Recommender System for Online Social Networks. *Future Gener. Comput. Syst.* **2019**, *93*, 914–923. [CrossRef]
16. Basic Machine Learning URL: https://machinelearningcoban.com/
17. Capdevila, Arias, M.; Arratia,A. GeoSRS: A Hybrid Social Recommender System for Geolocated Data. *Inf. Syst.* **2016**, *57*, 111–128.[CrossRef]
18. Tarus, J.K.; Niu, Z.; Yousif, A Hybrid Knowledge-Based Recommender System for e-Learning Based on Ontology andSequential Pattern Mining, *Future Gener. Comput. Syst.* **2017**, *72*, 37–48. [CrossRef]
19. Choi, S.-M.; Ko, S.-K.; Han, Y.-S. A Movie Recommendation Algorithm Based on Genre Correlations. *Expert Syst. Appl.* **2012**, *39*,8079–8085. [CrossRef]
20. Espy Yonder. Singular value decomposition to create a bench making data set from movielens data, 2014. URL:http://maheshakya.github.io/gsoc/2014/05/18/preparing-a-bench-marking-data-set-using-singula-value-decomposition-onmovielens-data.html.
21. James. Duling David. N. Langville Amy . D. Meyer Car Albright, Russell. Cox. Algorithms, initializations, and convergence for the nonnegative matrix factorization, 2014.URL http://www.dm.unibo.it/~simoncin/NMFInitAlgConv.pdf.
22. Bregt Verreet. The alternating least squares algorithm in recommenderlab. 2016,URL http://www.infofarm.be/articles/alternating-least-squares algorithmrecommenderlab.
23. Yehuda Koren. The bellkor solution to the netflix grand prize. 2009. URL http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.
24. Latent Factor Models. Matrix factorization methods, 2014. URL: http://recommender.no/algorithms/latent-factor-models-matrix-factorization-methods/
25. On the Difficulty of Evaluating Baselines, A Study on Recommender Systems - Steffen Rendle - 2019
26. Building a Movie Recommendation System using SVD algorithm - Asoke Nath - 2018
27. HybridSVD: When Collaborative Information is Not Enough - Evgeny Frolov - 2019
28. SVD++ Recommendation Algorithm Based on Backtracking - Shijie Wang – 2020