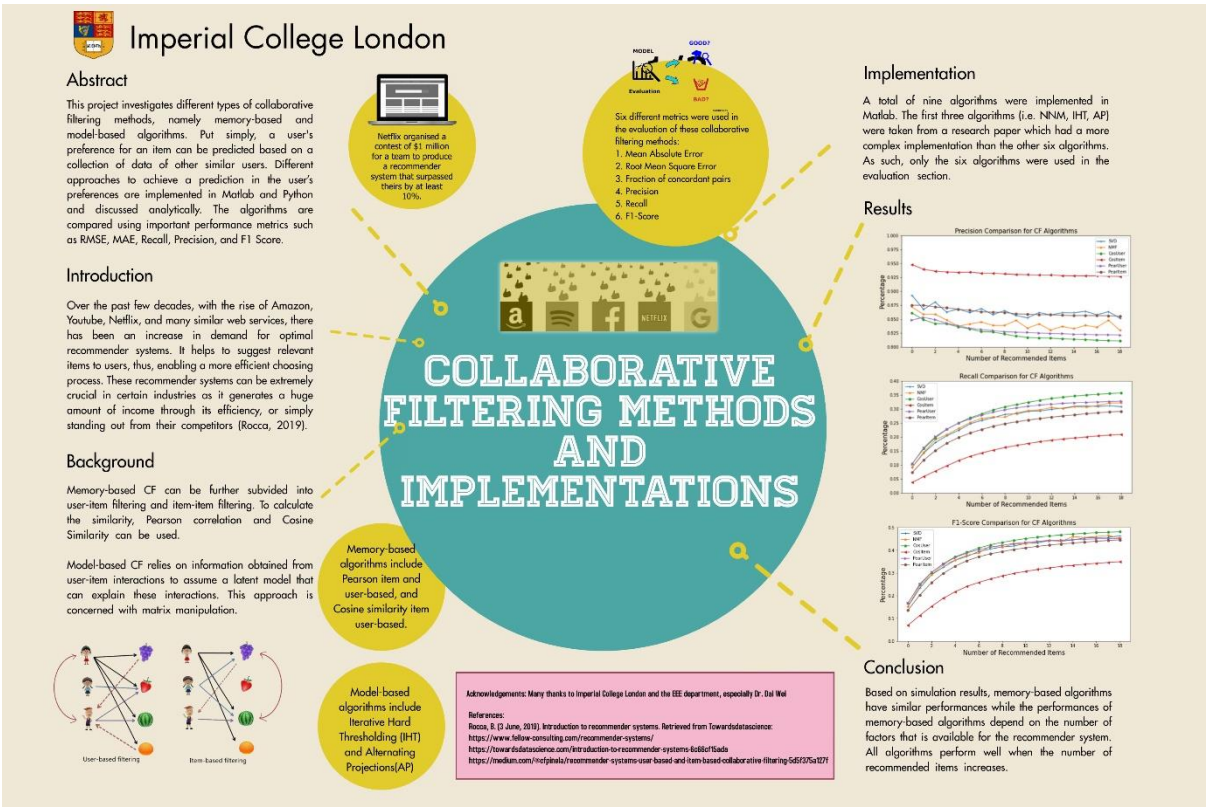
Poster containing the key points of project:



Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2021

Project Title:	**Collaborative Filtering Methods and Implementations**
Student:	**Jonathan Wong Shern Yang**
CID:	**01410438**
Course:	**EIE3**
Project Supervisor:	**Dr. Wei Dai**
Second Marker:	**Dr. Thomas Clarke**

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Table of Contents

1. **Abstract**

This project investigates different types of collaborative filtering methods – namely memory-based and model-based algorithms. Put simply, a user's preference for an item can be predicted based on a collection of data of other similar users. The collection of data consists of different users' preference on several items. This concept is defined and its application for recommender systems is introduced. In addition, different approaches to achieve a prediction in the user's preferences are implemented in Matlab and discussed analytically. To compare the performance of different algorithms on a real movie rating dataset by Netflix, we use the "Surprise" library in Python. The algorithms are compared using important performance metrics such as RMSE, MAE, Recall, Precision, and F1-Score.

2. **Introduction**

Over the past few decades, with the rise of Amazon, Youtube, Netflix, and many similar web services, there has been an increase in demand for optimal recommender systems. It helps to suggest relevant items to users, thus, enabling a more efficient choosing process. These recommender systems can be extremely crucial in certain industries as it generates a huge amount of income through its efficiency, or simply standing out from their competitors (Rocca, 2019). As proof of the importance of recommender systems, Netflix organised a contest with a prize of $1 million for a team that could produce a recommender system that surpassed theirs by a performance of 10% or more (Jackson, 2017). A recommender system has two broad categories – collaborative filtering methods and content-based methods. However, this project will only focus on the collaborative filtering methods. The goal of this project is to study "off-the-shelf" approaches to solve the user preference prediction problem. Comparing different algorithms used for collaborative filtering, this project aims to provide a numerical performance analysis in terms of accuracy and convergence to solve the low rank matrix completion problem.

Collaborative filtering is the idea that you will probably like things that, people with similar viewing habits, also like. It analyses the relationships between users and their interdependencies among products, thereby establishing new user-item associations (Koren, Bell, & Volinsky, Matrix Factorization Techniques For Recommender Systems). A reason for the preference for the collaborative filtering method is that while it is domain free, it can still address data aspects that are difficult to profile using content filtering (Koren, Bell, & Volinsky, Matrix Factorization Techniques For Recommender Systems). Even though it is more accurate than other content-based techniques, collaborative filtering has an issue of cold start problem (whereby the circumstances for recommender systems are not at its most optimal condition to provide the best possible results (Milankovich, 2015)) due to its inability to cater for the system's new products and users.

Collaborative filtering overcomes the problem of information overload by relying on past interactions between users and items to produce new recommendations, thereby storing these interactions in a "user-item interaction matrix" (Rocca, 2019). Through the matrix, the information collected is sufficient to detect similar users or to make accurate predictions for like-minded users.

The main advantage of collaborative filtering is that despite requiring no information about users or items, it can be used in many situations (Rocca, 2019). However, it only considers past interactions to make recommendations, which could lead to the "'cold start problem" (Gaspar, 2015). As a result of having too few interactions, these recommendation systems are unable to give good recommendations to new users as the circumstances are not optimal yet.

Under collaborative filtering, there are two categories – memory-based and model-based. For memory-based methods, it gathers information from the database and uses it to recommend to users. It also adjusts based on data changes, as a result causing a much greater computational time, depending on the data size. However, for the model-based method, it has a constant computational time as it does not adjust based on data changes (P. H. Aditya, 2016, p. 2). Some key findings of comparing memory-based and model-based collaborative filtering stated that memory-based methods were more superior in terms of precision and recall (P. H. Aditya, 2016, p. 1).

3. **Background**

In this section, collaborative filtering is broken down into the two main categories – Memory-based and Model-based. Collaborative filtering requires only the users' historical preference on a set of items, with the assumption that users, who agreed in the past, will likewise agree in future.

3.1. **Memory-based collaborative filtering**

The memory-based collaborative filtering can be further subdivided into user-item filtering and item-item filtering. User-item filtering stems from the idea that users who are similar to each other will have similar item preferences, while item-item filtering stems from the idea that users who liked an item, will also like another particular item (Rocca, 2019).

The most standard method of collaborative filtering is known as the nearest neighborhood algorithm (Luo, 2018), which is split between user-based and item-based. Having an n x m matrix of ratings, with user u_i and item p_j (i = 1,…,n ; j = 1,…,m). The aim is to predict the rating r_{ij} if the select user did not rate or watch the item j. To achieve this, the similarities between the select user and other users are calculated, identifying the top X similar users. Subsequently, a weighted average of the ratings from these X users is taken, having the similarities as weights. This gives the formula:

$$r_{ij} = \frac{\sum_k Similarities(u_i, u_k) \cdot r_{kj}}{number\ of\ ratings} \tag{1}$$

It is important to note that people can have differing measures of rating. For instance, some may give a generally higher score, while others may be stringent despite being satisfied with the items. Hence, to avoid this biasness, a subtraction of each user's average rating of all items can be done when computing the weighted average. Subsequently, added back for the selected user, as shown below.

$$r_{ij} = \overline{r}_i + \frac{\sum_k Similarities(u_i, u_k) \cdot (r_{kj} - \overline{r}_k)}{number\ of\ ratings} \tag{2}$$

To calculate the similarity, the Pearson Correlation and Cosine Similarity can be used.

$$Pearson\ Correlation(u_i, u_k) = \frac{\sum_j (r_{ij}, r_i) \cdot (r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \cdot \sum_j (r_{kj} - r_k)^2}} \tag{3}$$

$$Cosine\ Similarity(u_i, u_k) = \frac{r_i \cdot r_k}{|r_i||r_k|} = \frac{\sum_{j=1}^{m} r_{ij} r_{kj}}{\sqrt{\sum_{j=1}^{m} {r_{ij}}^2 \sum_{j=1}^{m} {r_{kj}}^2}} \tag{4}$$

Pearson correlation is a measure of the association between ordered pairs of continuous measurements from two groups, such as measuring pulse readings before and after exercise (R.H.Riffenburgh, 2012). Cosine similarity computes the similarity between all pairs of items (neo4j, n.d.).

Using the Pearson correlation and cosine similarity, we can find the most similar users to the selected user (nearest neighbour) and weight their ratings of an item, thus, being able to predict the rating of this item for the selected user.

An advantage of this method of collaborative filtering is that there is stability in the ratings for the given item, which will not have significant changes over time, unlike the preference of human beings. In addition, as there is no training involved (non-parametric machine learning approach), it is easy to implement. However, some drawbacks are that it is not able to handle sparsity well when there are few ratings for the item, and that it is hard to accommodate an increase in the number of users and items.

3.2. **Model-based collaborative filtering**

The model-based approach relies on information obtained from user-item interactions to assume a latent model that can explain these interactions (Rocca, 2019). This approach is concerned with matrix manipulation (Involving matrix completion algorithms which we will be implementing later). As such, it involves a step to reduce the large but sparse user-item matrix. To elaborate more on it, a basic idea of matrix completion and dimensionality reduction should be understood (Ajitsaria, n.d.).

3.2.1. **Matrix Completion**

A collaborative filtering scenario can be modelled as a matrix completion problem, whereby the input matrix of rows and columns represent users and items respectively, while the cells are the known preference statements for each user-item pair (Jannach & Zanker, 2018, p. 3). The ratings in the matrix can be explicit, such as having a score from 1-5 (e.g. strongly agree, agree, neutral, disagree, strongly disagree), or implicit, such as a preference of item purchases or store visits. By filling out these unknown entries and ranking them by their predicted values, we can make recommendations.

Table 1. User-Item Rating Matrix

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	2	3	4	?	
User 2		4	3	4	?
User 3		1	2	3	
User 4	1		3	3	2
User 5	1		1	3	

An example of such a matrix is shown in table 1, where matrix approaches are used to estimate or predict a rating for each unknown cell. The ultimate task in many of such applications is to filter and rank the items. While the ranking is determined based on the predicted rating, items that do not meet a minimum threshold will be filtered. The reason for using matrix completion for collaborative filtering is that the computational task is easily defined, while the genetic nature of the matrix problem allows people to design algorithms that can be used across many

application domains (Jannach & Zanker, 2018, p. 3). Moreover, several mathematics concepts for noise reduction and data analysis, such as singular value decomposition, can be applied directly on the data.

To understand matrix completion in a more mathematical sense, a matrix X_0 will assume to have size M x N, with rank R << M, N (Davenport & Romberg, 2016, January 24, p. 2). Hence, the matrix is written as:

$$X_0 \approx \sum_{k=1}^{R} \sigma_k u_k v_k{}^T \tag{5}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_R \geq 0$ and orthonormal vectors $u_1, u_2, \ldots, u_R \in R^M$ and $v_1, v_2, \ldots, v_R \in R^N$. Scalars $\{\sigma_k\}$ can be interpreted as the R largest singular values of matrix X_0, while $\{u_k\}\{v_k\}$ are the corresponding singular vectors. The collection of all these matrices form a union of subspaces in $R^{M \times N}$. The union of two subspaces is a subspace if one of the subspaces is contained within the other subspace. Each set of vectors $\{u_k\}$, $\{v_k\}$ defines an R-dimensional subspace, and $\{\sigma_k\}$ corresponds to the expansion in that subspace. Since $\{u_k\}$ and $\{v_k\}$ can vary, the union has indefinite number of subspaces. Hence, obtaining the degrees of freedom (number of remaining/free variables) in the singular value decomposition suggests that when R is small, R is much lesser than M and N used to specify a general matrix (Davenport & Romberg, 2016, January 24, p. 2). This suggests that despite the indefinite number of subspaces, it is possible to recover a low-rank matrix from relatively few measurements.

Instead of observing X_0 directly, we observe $y = X(A) + \varepsilon$, where a low rank matrix A is to be recovered and ε represents noise. However, we only get to observe a linear transformation of matrix A in X(A). This is a matrix version of linear regression (linear regression deals with vectors), where the idea of linear regression is to examine if a set of predictor variables give a good prediction in the outcome (dependent) variable, and to identify which variables are important predictors of the outcome variable (What is Linear Regression?, n.d.). The purpose of linear regression is to determine the strength of the predictors, forecasting an effect, and to forecast a trend. A is exactly or approximately low rank, while ε could be a noiseless or noisy case. The goal is to recover A from (y, X), and possibly design an appropriate X. Hence, the problem is to construct a linear map X (which is equivalent to Xi, where i = 1, …, n) to guarantee the stable recovery of low-rank matrices in the noiseless or noisy case with a minimum number of measurements.

3.2.2. **Dimensionality Reduction**

The purpose of dimensionality reduction is to help reduce either users or items to a lower dimensional space (as mentioned that model-based collaborative filtering utilises user-item interactions) (Wu, 2019).

If a matrix is mostly empty, using a method like matrix factorisation would help improve the performance of the algorithm by reducing its dimensions. Matrix factorisation can be understood as breaking down a large matrix into a product of two smaller matrices (Ajitsaria, n.d.). However, the product of matrix X and Y can only be obtained if the number of columns in X equals to the number of rows in Y.

The idea is that we can represent the users, items, and interactions between the user and item (by computing the dot product of the corresponding dense vectors in the space).

Consider an interaction matrix M of ratings where n users give ratings to m items. Hence, the matrix should be factorised:

$$M \approx X \cdot Y^T \tag{6}$$

where X represents the user matrix, with rows representing n users and Y represents the item matrix, with rows representing m items:

$$user_i \equiv X_i, \quad \forall_i \in \{1, \dots, n\} \tag{7}$$

$$item_j \equiv Y_i, \quad \forall_j \in \{1, \dots, m\} \tag{8}$$

Hence, conducting a search for matrices X and Y whose dot product gives the best approximation for the existing interactions. Let E be the ensemble of pairs (i, j) such that M_ij is set. As such, we want to find the appropriate X and Y such that the error from the rating reconstruction is minimised.

$$(X, Y) = argmin_{X,Y} \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2 \tag{9}$$

Also, it is important to find X and Y with the minimum power/energy and hence, can minimize their l2-norm by a regularization factor λ.

$$\begin{aligned}(X, Y) = argmin_{X,Y} & \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2 \\ & + \lambda(\sum_{i,k} (X_{ik})^2 + \sum_{j,k} (Y_{jk})^2)\end{aligned} \tag{10}$$

To solve this minimization problem, the gradient descent can be applied. The gradient does not have to be computed across all pairs in E for every step. Instead, it is possible to optimise them as a batch by looking at only a subset of these pairs. Moreover, the values in X and Y do not have to be updated simultaneously (Rocca, 2019).

Once the factorisation of the matrices is obtained, a new recommendation based of the factorised matrix can be made. A user vector is multiplied by any item vector to find out what the estimated rating is.

3.2.2.1 Singular Value Decomposition (SVD)

SVD is a type of dimensionality reduction which will be evaluated in section 5. The SVD is a common matrix factorization technique used for low-rank approximations (Sarwar, Riedl, & Karypis, 2002, p. 3). Given a m x n matrix A, with rank r, its SVD is defined as:

$$SVD(A) = U \times S \times V^T \tag{21}$$

Where U, S and V are of dimensions m x m, m x n, and n x n respectively. U and V are orthogonal matrices while matrix S is a diagonal matrix, also known as the singular matrix. As it only has r nonzero entries, it makes the effective dimensions of these three matrices m x r, r x r, and r x n respectively (Sarwar, Riedl, & Karypis, 2002, p. 9). The diagonal values of S have the property of $s_i > 0$ and $s_1 \geq s_2 \geq \ldots \geq s_r$. In addition, the r columns of U corresponding to the non-zero singular values span the column space, while the r columns of V span the row space of the matrix A.

The dimensionality reduction approach that the SVD applies can be very practical for the collaborative filtering process as it produces a set of uncorrelated eigenvectors, with each user and item represented by its corresponding eigenvector. Through the process of dimensionality reduction, customers who rate similar products can be mapped into the space of same eigenvectors.

3.2.2.2 Non-Negative Matrix Factorization (NMF)

The NMF is another type of dimensionality reduction which will be evaluated in section 5. The goal of NMF is to approximate a matrix (V) by the dot product of two arrays W and H (Gabrys, n.d.). If V has dimensions m x n and we wish to decompose it to j components, then W has n rows and j columns while H has j rows and m columns. As the name suggests, its values must be equal or greater than zero. To measure the difference between the reconstructed and initial array, we can use the Frobenius norm (Gabrys, n.d.):

$$A = V - W \cdot H \tag{32}$$

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a|_{ij}^2} \tag{13}$$

The algorithm can be illustrated with a grocery purchase example (Gabrys, n.d.). Let matrix W have values that indicating the weight of each item on a particular type of user, i.e. the higher the weight, the more "determined" the user (column) is by the variable (row).

Table 2. Grocery products and consumer types

	Fruit pickers	Bread eaters	Vegetable eaters
Vegetables	0	0.05	2.73
Fruits	1.84	0.16	0.45
Sweets	0.87	0	0
Bread	0	2.57	1.17
Coffee	0	0	0.57

A few examples are used to understand this matrix. Under fruit pickers, it is driven by two categories – fruits and sweets. However, bread eaters are driven by mostly by bread. If observed from a different perspective by the matrix's rows, it is important to note that coffee purchases contribute to only vegetable eaters, while for vegetable purchases, it has a higher weight towards vegetable eaters.

On the other hand, let matrix H represent the weights of how much a person uses a specific category.

Table 3. Consumer types and consumers

	John	Alice	Mary	Greg	Peter	Jennifer
Fruit pickers	1.02	1.27	0.52	0.25	0.84	0.87
Bread eaters	0	0.57	1.07	1.34	0.05	0.05
Vegetable eaters	0	0.33	0	0.31	0.74	0.65

For example, John is 100% in the fruit pickers cluster.

By multiplying W and H, we obtain V matrix approximation:

Table 4. Reconstructed matrix of grocery products and consumers

	John	**Alice**	**Mary**	Greg	Peter	Jennifer
Vegetables	0	**0.97**	**0.03**	0.97	2.09	1.88
Fruits	1.98	**2.80**	**1.20**	0.86	2.05	2.04
Sweets	1.04	**1.30**	**0.52**	0.24	0.84	0.85
Bread	0	**2.04**	**2.97**	4.05	0.97	0.98
Coffee	0	**0.18**	**0**	0.16	0.43	0.39

The reconstructed matrix is the result of collaborative filtering, where an assignment of values for previously unknown values (bolded) is obtained.

.

4. Implementation

4.1. Matrix Completion Algorithms

Three implementations were taken from the research paper named "An overview of low-rank matrix recovery from incomplete observations", by Mark A. Davenport and Justin Romberg – Low-Rank Recovery and Nuclear Norm Minimization, Iterative Hard Thresholding, and Alternating Projections. Only a few key parts of the code were used in this section to explain the implementation. The full code can be found in appendix 8.1. Before implementing these algorithms, synthetic data must be generated. A random Gaussian matrix of size M by N is generated, and the singular value decomposition is done to find the singular values and vectors. The R largest singular values are kept, and the rest are removed. This is done with the following Matlab code (the entire code is in appendix 8.1):

```
[matU, matS, matV] = svd(randn(M, N));
matS(R+1:end, R+1:end)=0; % make a lower rank matrix
matX0 = matU*matS*matV;
```

As explained in the low rank matrix recovery section, we will observe y = X(A) + ε, where ε represents noise and A : $R^{M \times N} \rightarrow R^{L}$ (R: real number) is a linear measurement operator that takes the standard inner products against L pre-defined M x N matrices $A_1, \dots, A_L$ (Davenport & Romberg, 2016, January 24, p. 3):

$$\begin{aligned} y_i = < X_0, A_i > + z_i &= trace(A_i{}^T X_0) + z_i \\ &= \sum_{m=1}^{M} \sum_{n=1}^{N} X_0[m,n] A_i[m,n] + z_i \end{aligned} \tag{44}$$

Using this matrix completion method from the research paper, it was implemented with the following code:

```
function tensorA = randomProjection(M, N, L)
% Input L, M, and N are defined in the paper in the first line of page 3.
% Output: A tensor of size L*M*N, i.e. L matrices of size M by N, each of
% which is one of Ai
tensorA = zeros(M, N, L);
for i=1:L
   tensorA(randi(M),randi(N),i) = 1;
end
```

Subsequently, we generate our observation and measurement of y, and contaminate it by a Gaussian noise, as explained in section [1] of the paper.

```
yVec = OperationAX0(tensorA,matX0)+sqrt(varNoise)*randn(L,1);
```

The function "OperationAX0" implements the operator "A" and the adjoint operator "A*" in equation (14) as follows

```
function vecAX0 = OperationAX0(tensorA, matX0)
```

```
[~, ~, L] = size(tensorA);
vecAX0 = zeros(L, 1);
for i=1:L
   vecAX0(i) = trace(matX0'*tensorA(:,:,i));
end
end

function matAadjoint = adjointOperation(tensorA, wVec)
[M, N, L] = size(tensorA);
matAadjoint = zeros(M, N);
for i=1:L
   matAadjoint = wVec(i)*tensorA(:,:,i) + matAadjoint;
end
end
```

Once this is completed, we assume that we have the measurement vector y and the operator A. Next, we want to find the matrix X in different ways, explained in three different sections of the paper.

4.1.1. **Low-Rank Recovery and Nuclear Norm Minimization (NNM)**

This algorithm works with indirect observations where we approximate $y \approx A(X_o)$, where we would want to work with:

$$minimise_x \; || \, y \; - \; A(X) \; ||_F^2 , rank(X) = R \tag{55}$$

We start by implementing the proximal operator in the following eq. (17) by using eq. (16) as follows:

$$\sigma'_k = \begin{cases} \sigma_k - \gamma, & \sigma_k \geq \gamma \\ 0, & \sigma_k \; < \gamma \end{cases} \tag{66}$$

$$prox_\gamma(Z) \; = \; argmin_X ||X \; - \; Z||_F^2 \; + \; \gamma\lambda||X||_* \tag{77}$$

```
function proxZ = proximalOperator (matZ, gamaPrime)

[mU, mS, mV] = svd(matZ);
mSprime = zeros(size(mS));
for k = 1:rank(mS)
   sigmaK = mS(k,k);
   if sigmaK < gamaPrime
      mSprime(k,k) = 0;
   else
      mSprime(k,k) = sigmaK - gamaPrime;
   end
end
proxZ = mU*mSprime*mV';
end
```

Subsequently, eq. (18) is implemented by the following code:

$$X_{k+1} = prox_{\gamma k}\,(X_k - {}_{\gamma k}A^*(A(X_k) - y)) \qquad (88)$$

```
X0byNNM = proximalOperator (X0byNNM - gamaa*adjointOperation(tensorA, OperationAX0(tensorA, X0byNNM)-yVec), gamaa);
```

It is necessary to note that we must iterate over this line up to a convergence. Thus, we put it in the for loop. By adjusting the number of iteration k, we make sure that the convergence happens.

To evaluate the performance, we propose to use the normalized mean square error of the original matrix and the estimated matrix, computed by the Frobenius norm as follows:

```
norm(matX0-X0byNNM,'fro')^2/(M*N*norm(matX0,'fro')^2)
```

We observe good results from this algorithm as shown from the very small error despite just running it through a few iterations.

4.1.2. **Iterative Hard Thresholding (IHT)**

The iterative hard thresholding algorithm computes the desired number of top R left and right singular vectors and singular values.

```
function matX = ProjectRank(matY, R)

[mU, mS, mV] = svd(matY);
mS(R+1:end, R+1:end)=0; % make a lower rank matrix
matX = mU*mS*mV';

end
```

By applying this function, we can estimate the matrix as by iterating over the following code:

```
matY = X0byIHT - gamaa*adjointOperation(tensorA, OperationAX0(tensorA, X0byIHT)-yVec);
X0byIHT = ProjectRank (matY, R);
```

When R is small when compared to M and N, we note that it takes much lesser time than to compute a full SVD (SVD: A matrix factorization method that generalizes the eigen decomposition of a square matrix to any matrix (Hinno, n.d.). SVD was explained in section 3.2.2.1), as there is a fast method for applying the matrices Y_{k+1} to the series of vectors (Davenport & Romberg, 2016, January 24, p. 6). Compared to the nuclear norm minimization, IHT has each of its iterations X_k that the algorithm produces has a prescribed rank. In addition, IHT has a massive storage reduction for large-scale applications as it reduces the storage required for X_k to about R(M + N) as compared to the required MN for a general M x N matrix.

4.1.3. **Alternating Projections (AP)**

In the AP, the alternating least square is targeted by applying the following equation:

$$R_{k+1} = argmin_R ||A(L_k R^T) - y||_2^2 \quad (99)$$

$$L_{k+1} = argmin_L ||A(L R_{k+1}^T) - y||_2^2 \quad (20)$$

Firstly, we decompose the main matrix X to a left L and a right R matrix. Subsequently, by fixing each of them, we try to estimate the other one and alternate between them up to a convergence. To implement the optimization problem, we use a built in Matlab function, called fminunc as follows:

```
estL = matUinit(:,1:R);
estR = matVinit(:,1:R);
options = optimoptions('fminunc','Display','none');
for k = 1:K
   funR = @(matR) norm(OperationAX0(tensorA, estL*matR')-yVec, 'fro')^2;
   [estR, ~] = fminunc(funR, estR, options);
   funL = @(matL) norm(OperationAX0(tensorA, matL*estR')-yVec, 'fro')^2;
   [estL, ~] = fminunc(funL, estL, options);
   X0byAP = estL*estR';
end
```

The advantage of the AP is that it relies on well-established algorithms in linear algebra, hence, making it simple and efficient for large-scale matrix factorizations (Davenport & Romberg, 2016, January 24, p. 6)). It also often outperforms NNM.

To initialize X for all the algorithms, a simple random Gaussian generation or the method explained in section 5 of the paper can be used, i.e. selecting some of the left eigenvectors for matrix L initialization.

4.2. **Memory-Based Algorithms**

For this implementation, we used the movie lens data set which can be downloaded at the following URL address: https://grouplens.org/datasets/movielens/. The full code can be found in appendix 8.2.

Reading the dataset:

```
ratingsData = readtable(fullfile('ml-100k','u.data'),'FileType','text',...
    'ReadVariableNames',false,'Format','%f%f%f%f');
ratingsData.Properties.VariableNames = {'user_id','movie_id','rating','timestamp'};
```

Constructing the rating matrix in which each entity is a rating of a user to a movie:

```
ratingMat = sparse(ratingsData.movie_id, ratingsData.user_id, ratingsData.rating);
```

The following command shows how many users and movies exists, respectively. We have 1682 users and 943 movies.

```
[numUsers, numMovies] = size(ratingMat);
```

The sparsity pattern of this matrix reads as follows:

```
figure
spy(ratingMat)
title('The known and missing ratings')
xlabel('User Indices')
ylabel('Movie Indices')
```

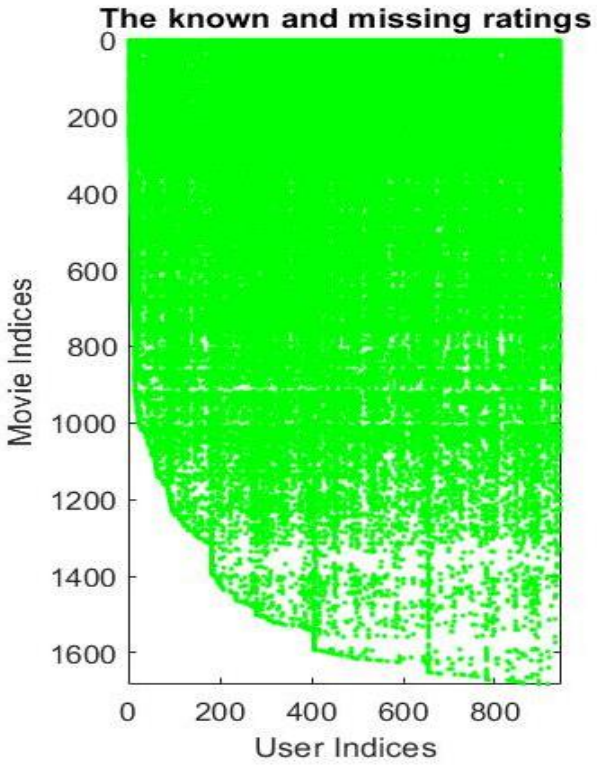


Figure 1. The known and missing ratings

As observed, the older movies with lower indexes have many ratings as there are likely to have more views from a greater number of users. For the higher indexes, the reverse is true.
As such, it is important to normalize the ratings by their means to scale all values in a fair manner:

```
meanRatings = sum(ratingMat,2)./sum(ratingMat~=0,2);
```

Subsequently, we compute the pair wise correlation of each two users to understand how their ideas are related to each other.

```
[i,j,v] = find(ratingMat);
meanCentered = sparse(i, j, v - meanRatings(i), numUsers, numMovies);
corrMat = corr(meanCentered', 'rows', 'pairwise');
```

We use this correlation matrix to predict a user's preferences based on other users' preferences. Prior to that, to reduce the computation overload, we can make the matrix even sparser by removing the diagonal elements, since the correlation of one user with himself/herself is not important and can be removed:

```
corrMat = sparse(corrMat - eye(numUsers));
```

To plot how the first user's idea is correlated with the first 10 users:

```
figure
plot(corrMat(1:10,1), '*')
```

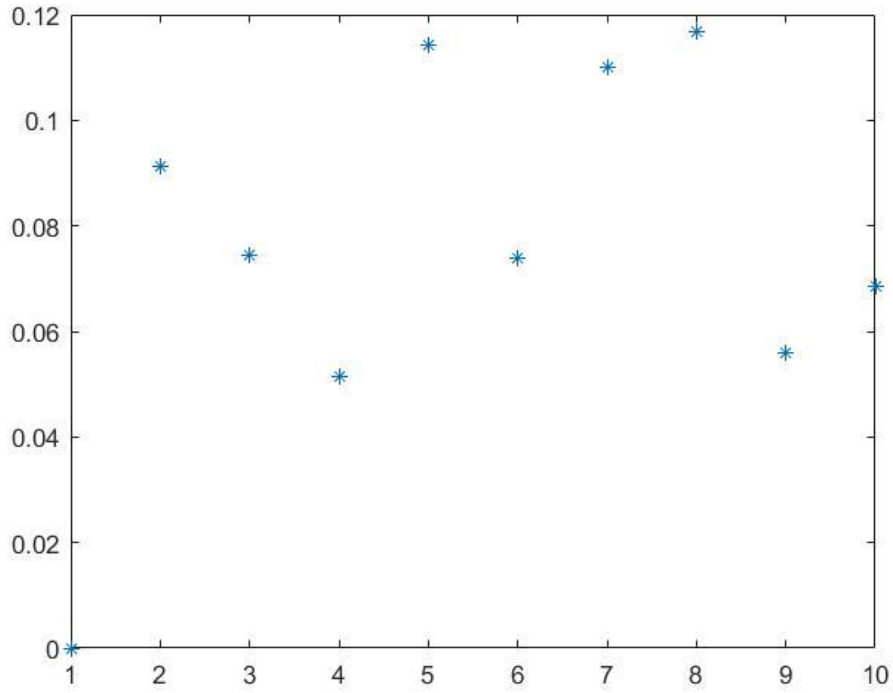


Figure 2. Correlation of the first user with first 10 users

From the graph obtained, the first user is highly correlated with user number 8 (among the first 10 users). We aim to find which user has the highest correlation with the first user throughout the entire data set by:

```
[highestCorrUser1, highestCorrInd] = max(corrMat(:,1));
```

The user 588 has the highest correlation with user 1 and the correlation value is about 0.25. This is a positive linear correlation. In opposite, we can find the highest negative correlation as follows:

```
[lowesCorrUser1, lowestCorrInd] = min(corrMat(:,1));
```

The results identify that it is user 163. Moreover, it shows that if this user likes a movie, there is a high probability that the first user dislikes that movie.

5. Results and Evaluation

A consensus has not been reached on what aspects should be evaluated for collaborative filtering algorithms. It seems that the most common measurement metrics revolve around evaluating the algorithms' accuracy and precision (Cacheda, Fernandez, Carneiro, & Formoso, 2011, p. 6). To compare the accuracy, we define and use six different metrics, namely mean absolute error, root mean square, fraction of concordant pairs, precision, recall and f1 score. In addition, the Netflix movie ratings dataset is used. The full code is in appendix 8.4.

5.1. Mean Absolute Error

Mean absolute error (MAE) – Measures the difference between the rating produced by the system and the actual rating as an absolute value. It is computed using a formula which takes all the ratings available:

$$|\bar{E}| = \frac{\sum_i^N |p_i - v_i|}{N} \tag{21}$$

The simplicity of its calculation has made this one of the most popular metrics when evaluating recommender systems. The results are in table 6 of section 5.3.

5.2. Root Mean Square Error

Root mean squared error (RMSE) – Is related to MAE, except that it places a greater emphasis on larger errors by making the error difference more pronounced:

$$|\bar{E}| = \sqrt{\frac{\sum_i^N (p_i - v_i)^2}{N}} \tag{22}$$

The main idea of using this metric is that errors can have the greatest impact on the user's decision. The results are in table 6 of section 5.3.

So far, we used the default value of 100 for the number of factors. A table of varying factors with its error was constructed:

Table 5. Varying numbers of factors for the SVD algorithm and corresponding errors

k	error
5	0.938425
10	0.935949
25	0.935627
50	0.939623
75	0.935788
100	0.940412

As observed, despite the higher the number of factors, its error increases only slightly. Furthermore, we can depict how the errors of the test set varies with the number of factors.

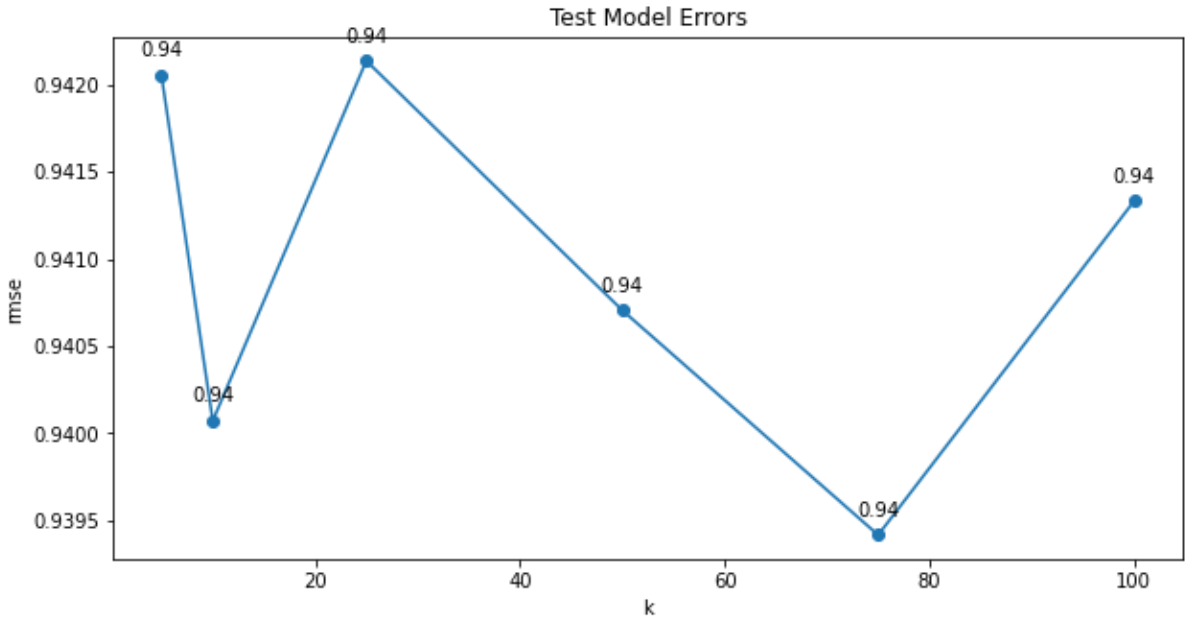


Figure 3. Varying number of factors and its corresponding errors

5.3. **Fraction of concordant pairs**

Fraction of concordant pairs (FCP) – Is related to RMSE. The RMSE, has several disadvantages because it assumes numerical rating values. It is not able to express rating scales that vary among different users (Koren & Sill, Collaborative Filtering on Ordinal User Feedback, 2013). In addition, if ratings are ordinal, we will not be able to use RMSE. Hence, the use of a ranking-oriented metric such as FCP will be useful. Given a test set R, the number of concordant pairs for user u is defined by counting ranked correctly by the rating predictor $\hat{r}_u$.

$$n_c^u = \left|\{(i,j) \mid \hat{r}_{ui} > \hat{r}_{uj} \; and \; r_{ui} > r_{uj}\}\right| \tag{23}$$

We count the discordant pairs n_c^u for user u and after summing over all users, we define $n_c = \sum_u n_c^u$ and $n_d = \sum_u n_d^u$. FCP serves as a metric to measure the proportion of well ranked item pairs.

$$FCP = \frac{n_c}{n_c + n_d} \tag{24}$$

Table 6. Results of metrics applied to the algorithms

	RMSE	MAE	FCP
SVD	0.93	0.74	0.70
NMF	0.96	0.76	0.69
CosItem	1.03	0.82	0.58
CosUser	1.02	0.81	0.71
PearItem	1.00	0.79	0.67
PearUser	1.01	0.80	0.72

Where CosItem, CosUser, PearItem, and PearUser refer to the cosine similarity of items, cosine similarity of users, Pearson correlation of items, and Pearson correlation of users, respectively. By implementing these 6 common collaborative filtering methods, we can compare the results of its RMSE, MAE, and FCP values, as displayed in Table 6. The code used to calculate these values can be found in appendix 8.3, where the implementation and how the figures were obtained are explained.

5.4. **Precision**

Three other important performance metrics are recall, precision, and f1-score that we will discuss next. By implementing these three measures on all six algorithms, we found that as the number of recommended items increases, the algorithms output a better performance. The general shape of the curves is similar for all algorithms. The following figure illustrates how these three measures change for the collaborative filtering implemented by Pearson correlation algorithm.

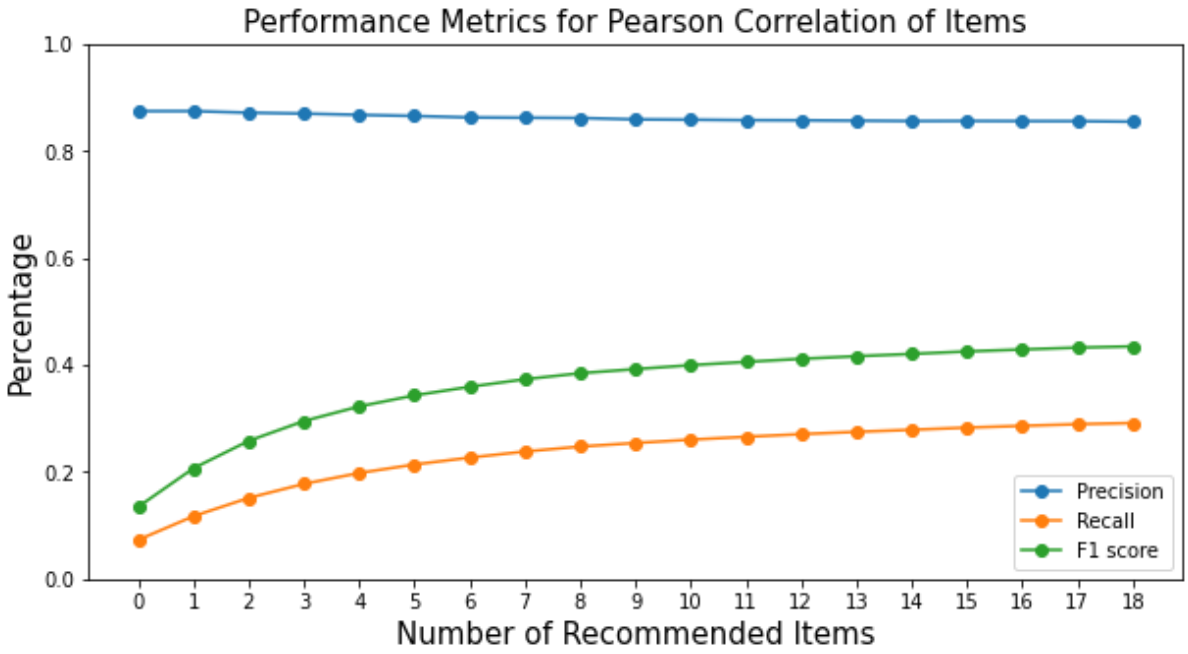


Figure 4. Precision, recall and F1-score implemented by Pearson correlation algorithm

We compared each metric for all collaborative filtering algorithms separately as follows.

Precision is the ratio of relevant items to recommended items. It can be defined as the true positives divided by the number of true positives plus the number of false positives. True positives are cases where the model identifies correctly as positive, and false positives are cases where the model identifies incorrectly as negative (Beyond Accuracy: Precision and Recall, 2018). It is expressed by the following formula:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{25}$$

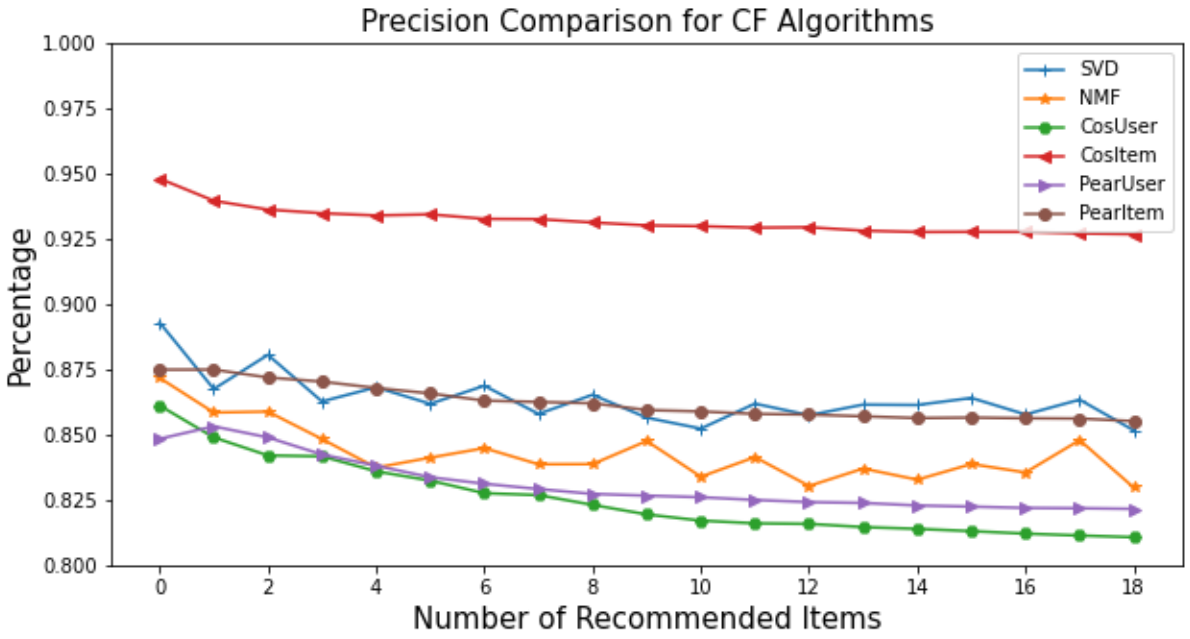


Figure 5. Precision comparison for CF algorithms

As illustrated from this figure, it is evident that the cosine similarity of items performs the best among the algorithms while the cosine similarity of users performs the least well, in terms of precision.

5.5. **Recall**

Recall is the proportion of recommended relevant items over the total number of relevant items (Cacheda, Fernandez, Carneiro, & Formoso, 2011, p. 9). It can be defined as the true positives divided by the number of true positives plus the number of false negatives. False negatives are cases where the model identifies incorrectly as positive (Beyond Accuracy: Precision and Recall, 2018). It is expressed by the following formula:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \qquad (26)$$

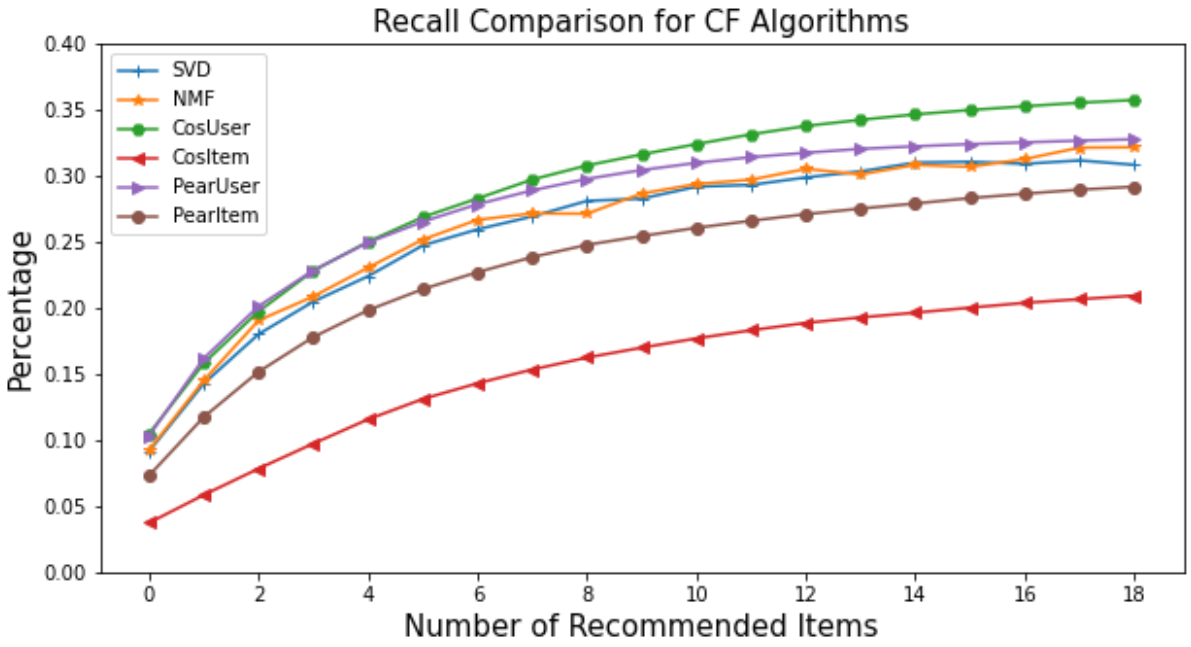


Figure 6. Recall comparison for CF algorithms

On the other hand, when doing a recall comparison, the results seem to be the opposite of the recall results. The cosine similarity of items produces the worst performance, while the cosine similarity of the users performs the best. This is in line with a concept discussed earlier, where precision and recall have an inverse relationship.

5.6. **F1-Score**

Most systems ideally want high precision and recall values. However, as precision and recall are inversely related, a metric F1 provides a more comprehensive analysis since it includes the effect of both precision and recall, thus giving a better measure of the incorrectly classified cases:

$$F1 = \frac{2Precision \cdot Recall}{Precision + Recall} \tag{27}$$

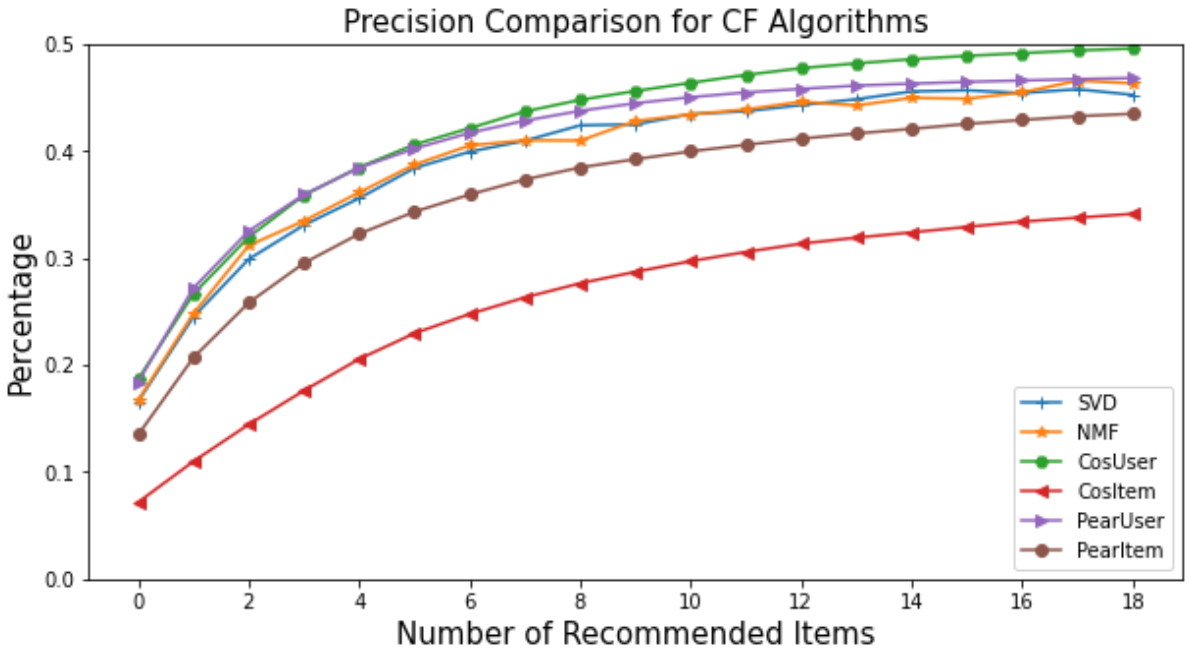


Figure 7. Precision comparison for CF algorithms

The cosine similarity of users performs the best while the other algorithms (except the cosine similarity of items) have relatively close performances. For this dataset, it is very clear that the similarity of items algorithm is not a good algorithm. Another important note is that the SVD and NMF have very similar performances as this is probably because both are based on matrix manipulation and that they are part of the model-based collaborative filtering.

6. Conclusions and Further Work

In this project, an investigation on recommendation system based on collaborative filtering methods was conducted. The memory based and model-based methods have been studied in general, along with the required mathematical tools to approach them. It reviewed the analytical perspective of these algorithms and implemented them with examples in Matlab and Python scripts in section 4 and 5. Considering simulation results, the model-based methods have similar performances while the performances of memory-based algorithms depend on the number of factors that is available for the recommender system in section 5. All algorithms perform well when the number of recommended items increases.

The ratings can come in many forms, including thumbs up/down, A-F grades, etc. Furthermore, the algorithms treat the ratings as numeric or binary, which might be convenient to model, but might not necessarily apply to all cases. For instance, it may be hard to model an online shopper's preference based off his/her shopping cart using a numeric approach, as there is a range of possible actions that the shopper can choose.

In a future study, more approaches to cater for numeric and non-numeric users should be explored. One of such examples is to take user feedback – OrdRec model, which treats user ratings as ordinal. The framework converts the model to a probability distribution over the ordinary set of ratings.

This study used a few algorithms (NNM, IHT, AP, etc.) in the implementation of memory-based and model-based collaborative filtering. Six different performance metrics were used in the evaluation. In future studies, more algorithms and different types of performance metrics could be used to provide a more comprehensive analysis. Such metrics could include diversity and novelty, which was too complex to include within the scope of this project.

As observed from the SVD computations, it generally leads to very fast performances. Future work can be done to understand why SVD works better for certain recommender systems and getting worse for others. In addition, SVD can be used in other ways for a recommender system. For instance, using it to identify significant products, which would help in bootstrapping the recommender system.

Lastly, besides the three algorithms implemented for matrix completion in section 4.1, there are many other methods that exist for matrix completion, where there is no efficient implementation for big data. The project started off with implementing the three algorithms as my supervisor was helping me get a better understanding of the topic. However, after researching more into the topic, I realized that these three algorithms were part of a paper which is analytical and research-based. Hence, I did not have enough time to implement the three algorithms efficiently in Matlab. The code needs to be refined many times to be able to exist in the market of machine learning and recommender systems, just like the other six algorithms used for comparison in section 5.

7. **Bibliography**

Ajitsaria, A. (n.d.). *Build a Recommendation Engine With Collaborative Filtering*. Retrieved from realpython: https://realpython.com/build-recommendation-engine-collaborative-filtering/

Beyond Accuracy: Precision and Recall. (4 3, 2018). Retrieved from towardsdatascience: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

Cacheda, F., Fernandez, D., Carneiro, V., & Formoso, V. (2011). *Comparison of Collaborative Filtering Algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems.* Researchgate.

Davenport, M. A., & Romberg, J. (2016, January 24). *An overview of low-rank matrix recovery from incomplete observations.*

Gabrys, P. (n.d.). *Non-negative matrix factorization for recommendation systems*. Retrieved from Medium: https://medium.com/logicai/non-negative-matrix-factorization-for-recommendation-systems-985ca8d5c16c

Gaspar, H. (14 July, 2015). *The Cold Start Problem for Recommender Systems*. Retrieved from Yuspify: https://yuspify.com/blog/cold-start-problem-recommender-systems/

Hinno, R. (n.d.). *Simple SVD algorithms*. Retrieved from towardsdatascience: https://towardsdatascience.com/simple-svd-algorithms-13291ad2eef2

Jackson, D. (7 August, 2017). *The Netflix Prize: How a $1 Million Contest Changed Binge-Watching Forever*. Retrieved from Thrillist: https://www.thrillist.com/entertainment/nation/the-netflix-prize

Jannach, D., & Zanker, M. (2018). *Collaborative Filtering: Matrix Completion and Session-Based Recommendation Tasks.*

Koren, Y., & Sill, J. (2013). *Collaborative Filtering on Ordinal User Feedback.*

Koren, Y., & Sill, J. (2013). *Collaborative Filtering on Ordinal User Feedback.*

Koren, Y., Bell, R., & Volinsky, C. (n.d.). *Matrix Factorization Techniques For Recommender Systems.* Retrieved from datajobs: https://datajobs.com/data-science-repo/Recommender-Systems-%5bNetflix%5d.pdf

Luo, S. (10 December, 2018). *Introduction to Recommender System*. Retrieved from towardsdatascience: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

Milankovich, M. (14 July, 2015). *The Cold Start Problem for Recommender Systems*. Retrieved from Medium: https://medium.com/yusp/the-cold-start-problem-for-recommender-systems-89a76505a7

neo4j. (n.d.). *The Cosine Similarity algorithm*. Retrieved from neo4j: https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/cosine/

P. H. Aditya, I. B. (2016). A Comparative Analysis of Memory-based and Model-based Collaborative Filtering on the Implementation of Recommender System for E-commerce in Indonesia : A Case Study PT X., (p. 6).

R.H.Riffenburgh. (2012). *Statistics in Medicine* (Third ed.). San Diego, California, USA.

Rocca, B. (3 June, 2019). *Introduction to recommender systems*. Retrieved from Towardsdatascience: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

Sarwar, B., Riedl, J., & Karypis, G. (2002). *Incremental singular value decomposition algorithms for highly scalable recommender systems.* Researchgate.

What is Linear Regression? (n.d.). Retrieved from statisticssolutions: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-linear-regression/

Wu, J. (28 May, 2019). *Improving Collaborative Filtering with Dimensionality Reduction*. Retrieved from Medium: https://medium.com/@jwu2/improving-collaborative-filtering-with-dimensionality-reduction-a99d08585dab#:~:text=In%20the%20case%20of%20user,also%20yields%20improvement%20in%20accuracy.

Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer, 42(8), pp.30-37.

Jain, Prateek, Praneeth Netrapalli, and Sujay Sanghavi. "Low-rank matrix completion using alternating minimization." *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013.
Jain, Prateek, Praneeth Netrapalli, and Sujay Sanghavi. "Low-rank matrix completion using alternating minimization." *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013.

8. Appendices

8.1. The entire .m file implementing algorithms 3.2-3.4 from research paper "An overview of low-rank matrix recovery from incomplete observations", by Mark A. Davenport and Justin Romberg

```
clear
clc
close all

M = 10; % dim of matrix X0
N = 30; % dim of matrix X0
R = 10; % rank of the observation matrix X0
L = 50; % dim of linear measurement operator A
varNoise = 0.01; % noise variance


%% Synthetic Data Generation
[matU, matS, matV] = svd(randn(M, N));
matS(R+1:end, R+1:end)=0; % make a lower rank matrix
matX0 = matU*matS*matV';
tensorA = randomProjection(M, N, L);
yVec = OperationAX0(tensorA, matX0)+sqrt(varNoise)*randn(L,1);
%

%% 3.2 Low-rank recovery and nuclear norm minimization

%Initialization by the first R singular vectors!
[matUinit, matSinit, matVinit] = svd(adjointOperation(tensorA, yVec));
matSinit(R+1:end, R+1:end)=0;
matXinit = matUinit*matSinit*matVinit';

% Here, we implement (9) and finds the solution for (7).
gamaa = 0.001; % to initialize gamma in (9)
K = 10; % Number of iterations to converge
X0byNNM = matXinit; % finding X0 by using nuclear norm minimization of 3.2
for k=1:K
   X0byNNM = proximalOperator (X0byNNM - gamaa*adjointOperation(tensorA, OperationAX0(tensorA, X0byNNM)-yVec), gamaa);
%          estX0 = proximalOperator (estX0 - gamaa*adjointOperation(tensorA, OperationAX0(tensorA, estX0)-yVec) , gamaa);
   gamaa = gamaa/100; %we can change it later!
   sprintf('Performance 3.2: Normalized error at this iteration is %d',norm(matX0-X0byNNM,'fro')^2/(M*N*norm(matX0,'fro')^2))
end

%% 3.3 Iterative hard thresholding

X0byIHT = matXinit; % finding X0 by using Iterative Hard Thresholding of 3.3
```

```
gamaa = 0.0001;
for k=1:K
   matY = X0byIHT - gamaa*adjointOperation(tensorA, OperationAX0(tensorA, X0byIHT)-
yVec);
   X0byIHT = ProjectRank (matY, R);
   gamaa = gamaa/100; %we can change it later!
   sprintf('Performance 3.3: Normalized error at this iteration is %d',norm(matX0-
X0byIHT,'fro')^2/(M*N*norm(matX0,'fro')^2))
end

%% 3.4 Alternating projections
X0byAP = matXinit;
estL = matUinit(:,1:R);
estR = matVinit(:,1:R);
options = optimoptions('fminunc','Display','none');
for k = 1:K
   funR = @(matR) norm(OperationAX0(tensorA, estL*matR')-yVec, 'fro')^2;
   [estR, ~] = fminunc(funR, estR, options);
   funL = @(matL) norm(OperationAX0(tensorA, matL*estR')-yVec, 'fro')^2;
   [estL, ~] = fminunc(funL, estL, options);
   X0byAP = estL*estR';
   sprintf('Performance 3.4: Normalized error at this iteration is %d',norm(matX0-
X0byAP,'fro')^2/(M*N*norm(matX0,'fro')^2))
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Functions%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3290097/
%https://uwspace.uwaterloo.ca/bitstream/handle/10012/13220/karimi_amir-
hossein.pdf?isAllowed=y&sequence=3
%https://www.researchgate.net/profile/Matej-
Kristan/publication/228944345_Efficient_Dimensionality_Reduction_Using_Random_Projec
tion/links/004635177a9733665e000000/Efficient-Dimensionality-Reduction-Using-Random-
Projection.pdf

function matX = ProjectRank(matY, R)
% This function implements the last equation in page 5, i.e. projecting
% onto the set of rank R matrices

[mU, mS, mV] = svd(matY);
mS(R+1:end, R+1:end)=0; % make a lower rank matrix
matX = mU*mS*mV';

end

function proxZ = proximalOperator (matZ, gamaPrime)
```

```
% Implementing (8) of the paper
% input coefficient gamaPrime = gama*lambda or the regularization parameter
% of the optimization problem in (8) (also, we can assume it as gama in (5)

[mU, mS, mV] = svd(matZ);
mSprime = zeros(size(mS));
for k = 1:rank(mS)
   sigmaK = mS(k,k);
   if sigmaK < gamaPrime
      mSprime(k,k) = 0;
   else
      mSprime(k,k) = sigmaK - gamaPrime;
   end
end
proxZ = mU*mSprime*mV;
end

function vecAX0 = OperationAX0(tensorA, matX0)
% To find the operation of \cal(A)(X0) at the bottom of the page 2 of the
% paper
[~, ~, L] = size(tensorA);
vecAX0 = zeros(L, 1);
for i=1:L
   vecAX0(i) = trace(matX0'*tensorA(:,:,i));
end
end

function matAadjoint = adjointOperation(tensorA, wVec)
% This function aims to find the adjoint of the operator A, by following
% the equation exactly below the (1) of the paper
[M, N, L] = size(tensorA);
matAadjoint = zeros(M, N);
for i=1:L
   matAadjoint = wVec(i)*tensorA(:,:,i) + matAadjoint;
end
end

function tensorA = randomProjection(M, N, L)
% Input L,M, and N are defined in the paper in the first line of page 3.
% Output: A tensor of size L*M*N, i.e. L matrices of size M by N, each of
%         which is one of Ai

% Using the help of the following pages
%
https://en.wikipedia.org/wiki/Random_projection#:~:text=In%20mathematics%20and%20statistics%2C%20random,when%20compared%20to%20other%20methods.
%      https://www.mathworks.com/help/stats/work-with-multinomial-probability-distribution-objects.html

rng('default')  % For reproducibility
```

```
pd = makedist('Multinomial','Probabilities',[1/6 2/3 1/6]);
multinomialRandom = random(pd, 1, M*N*L);
tensorA = reshape(sqrt(3)*(multinomialRandom-2), M,N,L);
end
```

8.2. **The entire .m file implementing movie recommendation memory-based algorithm**

```
clear
clc
close all
ratingsData = readtable(fullfile('ml-100k','u.data'),'FileType','text',...
    'ReadVariableNames',false,'Format','%f%f%f%f');
ratingsData.Properties.VariableNames = {'user_id','movie_id','rating','timestamp'};

ratingMat = sparse(ratingsData.movie_id, ratingsData.user_id, ratingsData.rating);

[numUsers, numMovies] = size(ratingMat);

figure
spy(ratingMat, 'g')
title('The known and missing ratings')
xlabel('User Indices')
ylabel('Movie Indices')

meanRatings = sum(ratingMat,2)./sum(ratingMat~=0,2);

[i,j,v] = find(ratingMat);
meanCentered = sparse(i, j, v - meanRatings(i), numUsers, numMovies);
corrMat = corr(meanCentered', 'rows', 'pairwise');

corrMat = sparse(corrMat - eye(numUsers));

figure
plot(corrMat(1:10,1), '*')

[highestCorrUser1, highestCorrInd] = max(corrMat(:,1));

[lowesCorrUser1, lowestCorrInd] = min(corrMat(:,1));
```

8.3. **Collaborative Filtering Algorithms and Comparison**

We implemented some of the famous collaborative filtering methods and compared the results. The dataset is from Netflix movie database which has movie ratings for a lot of users. We have installed and used the "surprise" library which had lots of commands for the recommender system implementations.

Firstly, we imported the required methods from the library

```
from surprise import SVD, NMF, Dataset, accuracy, KNNBasic
from surprise.model_selection import train_test_split
```

Subsequently, we use the movie lens dataset from Netflix

```
data = Dataset.load_builtin('ml-100k')
```

25% of the data is used for testing the algorithm

```
trainset, testset = train_test_split(data, test_size=.25)
```

The first recommender system is initiated based on SVD. It is trained and tested by the train and test datasets respectively:

```
algoSVD = SVD()
algoSVD.fit(trainset)
predictSVD = algoSVD.test(testset)
```

The second recommender system is based on matrix factorization.

```
algoMF =NMF()
algoMF.fit(trainset)
predictMF = algoMF.test(testset)
```

The third recommender system is based on the cosine similarity of the items.

```
sim_options = {'name': 'cosine',
               'user_based': False  # compute  similarities between items
               }
algoCosItem = KNNBasic(sim_options=sim_options)
algoCosItem.fit(trainset)
predictCosItem = algoCosItem.test(testset)
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

The fourth recommender system is based on the cosine similarity of the users.

```
sim_options = {'name': 'cosine',
               'user_based': True  # compute  similarities between users
               }
algoCosUser = KNNBasic(sim_options=sim_options)
algoCosUser.fit(trainset)
predictCosUser = algoCosUser.test(testset)
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

The fifth recommender system is based on the Pearson correlation of items.

```
sim_options = {'name': 'pearson_baseline',
               'user_based': False  # compute  similarities between users
```

```
            }
algoPearItem = KNNBasic(sim_options=sim_options)
algoPearItem.fit(trainset)
predictPearItem = algoPearItem.test(testset)
```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.

The sixth recommender system is based on the Pearson correlation of users.
```
sim_options = {'name': 'pearson_baseline',
            'user_based': True  # compute  similarities between users
            }
algoPearUser = KNNBasic(sim_options=sim_options)
algoPearUser.fit(trainset)
predictPearUser = algoPearUser.test(testset)
```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.

Root Mean Square Error
```
rmseSVD = accuracy.rmse(predictSVD, verbose=False)
print("RMSE of the SVD algorithm is:", rmseSVD)
rmseMF= accuracy.rmse(predictMF, verbose=False)
print("RMSE of the MF algorithm is:",rmseMF)
rmseCosItem = accuracy.rmse(predictCosItem, verbose=False)
print("RMSE of the items Cosine similarity is:",rmseCosItem)
rmseCosUser = accuracy.rmse(predictCosUser, verbose=False)
print("RMSE of the users Cosine similarity is:", rmseCosUser)
rmsePearItem = accuracy.rmse(predictPearItem, verbose=False)
print("RMSE of the items Pearson similarity is:", rmsePearItem)
rmsePearUser = accuracy.rmse(predictPearUser, verbose=False)
print("RMSE of the users Pearson similarity is:", rmsePearUser)
```
RMSE of the SVD algorithm is: 0.9352706734305519
RMSE of the MF algorithm is: 0.9650534757641753
RMSE of the items Cosine similarity is: 1.0296746645085195
RMSE of the users Cosine similarity is: 1.0193070208149344
RMSE of the items Pearson similarity is: 0.9971265769634031
RMSE of the users Pearson similarity is: 1.005160708414962
Mean Absolute Error
```
maeSVD = accuracy.mae(predictSVD, verbose=False)
print("MAE of the SVD algorithm is:", maeSVD)
maeMF = accuracy.mae(predictMF, verbose=False)
print("MAE of the MF algorithm is:", maeMF)
maeCosItem = accuracy.mae(predictCosItem, verbose=False)
print("MAE of the items Cosine similarity is:", maeCosItem)
maeCosUser = accuracy.mae(predictCosUser, verbose=False)
print("MAE of the users Cosine similarity is:", maeCosUser)
maePearItem = accuracy.mae(predictPearItem, verbose=False)
print("MAE of the items Pearson similarity is:", maePearItem)
```

```
maePearUser = accuracy.mae(predictPearUser, verbose=False)
print("MAE of the users Pearson similarity is:", maePearUser)
```
```
MAE of the SVD algorithm is: 0.7380397290829462
MAE of the MF algorithm is: 0.7595765205454458
MAE of the items Cosine similarity is: 0.8154083295637806
MAE of the users Cosine similarity is: 0.8089432202421933
MAE of the items Pearson similarity is: 0.7854806468804182
MAE of the users Pearson similarity is: 0.7956374079240621
```
```
Fraction of Concordant Pairs
fcpSVD = accuracy.fcp(predictSVD, verbose=False)
print("FCP of the SVD algorithm is:", fcpSVD)
fcpMF = accuracy.fcp(predictMF, verbose=False)
print("FCP of the MF algorithm is:", fcpMF)
fcpCosItem = accuracy.fcp(predictCosItem, verbose=False)
print("FCP of the items Cosine similarity is:", fcpCosItem)
fcpCosUser = accuracy.fcp(predictCosUser, verbose=False)
print("FCP of the users Cosine similarity is:", fcpCosUser)
fcpPearItem = accuracy.fcp(predictPearItem, verbose=False)
print("FCP of the items Pearson similarity is:", fcpPearItem)
fcpPearUser = accuracy.fcp(predictPearUser, verbose=False)
print("FCP of the users Pearson similarity is:", fcpPearUser)
```
```
FCP of the SVD algorithm is: 0.6999947627504953
FCP of the MF algorithm is: 0.6902632512513237
FCP of the items Cosine similarity is: 0.58454167134837
FCP of the users Cosine similarity is: 0.7077702572230119
FCP of the items Pearson similarity is: 0.6683596436910746
FCP of the users Pearson similarity is: 0.7165899959979004
```

8.4. **Tuning Parameters and Evaluating SVD**

All the algorithms mentioned above can have their parameters tuned. As an example, we decided to focus on just the SVD algorithm. One of the important parameters in the SVD algorithm is the number of factors. Initially a value of 100, we changed the number of factors and recorded the errors associated with it.

```
#Importing more libraries
import os
import pandas as pd
import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
k_factors = [5, 10, 25, 50, 75, 100]
test_rmse = []
algo = SVD()
for k in k_factors:
    algo = SVD(n_factors=k, verbose=False)
    algo.fit(trainset)
    predictions = algo.test(testset)
    error = accuracy.rmse(predictions, verbose = False)
    test_rmse.append(error)
error_data = {'k': k_factors, 'error': test_rmse}
pd.DataFrame(error_data)
```

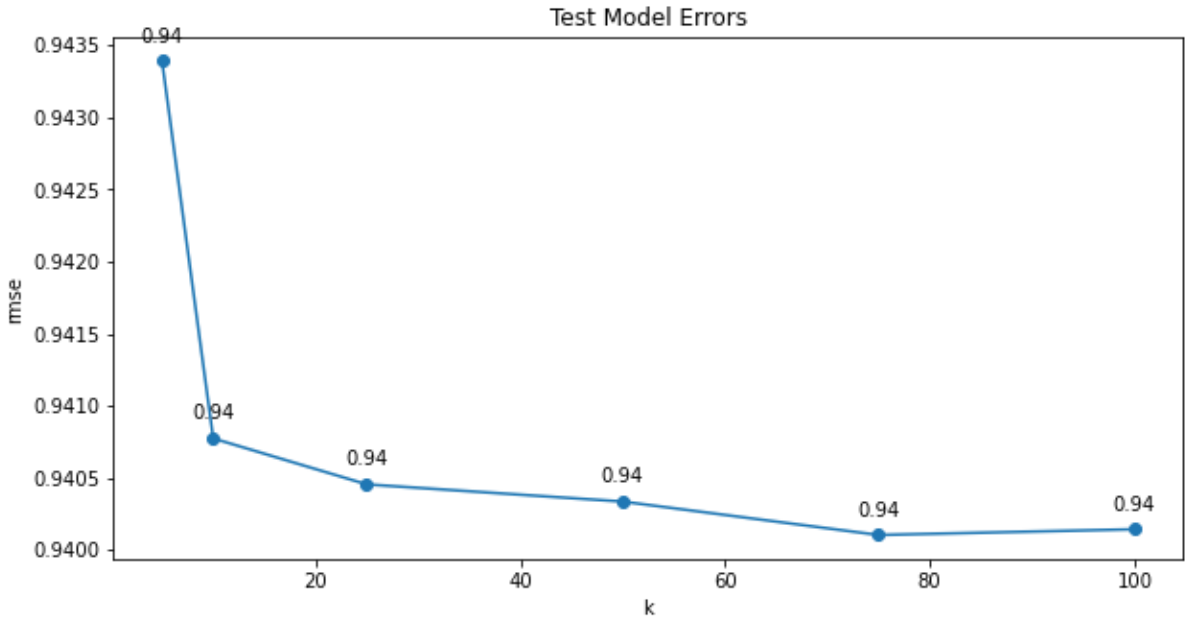
k	error
5	0.938425
10	0.935949
25	0.935627
50	0.939623
75	0.935788
100	0.940412

Also, we can depict how the errors of the test set variates with the number of factors.

```
def plot_model_rmse(xs, ys, title, x_label, y_label):
    # Set up the matplotlib figure
    fig, ax = plt.subplots(figsize = (10, 5))
    ax.plot(xs, ys, marker = 'o')
    for x,y in zip(xs,ys):
        label = "{:.2f}".format(y)
        plt.annotate(label, (x,y), textcoords="offset points", xytext=(0,10), ha='center')

    plt.title(title, fontsize = 12)
    plt.xlabel(x_label, fontsize = 10)
```

```
    plt.ylabel(y_label, fontsize = 10)
    plt.draw()
plot_model_rmse(error_data['k'], error_data['error'], 'Test Model Errors', 'k', 'rmse')
```



Recall, Precision, and F1-Score

The other three important performance measure that we can apply, are recall, precision, and f1-score.

```
# Return precision and recall at k metrics for each user
def precision_recall_at_k(predictions, k = 10, threshold = 3.5):

    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():

        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Number of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold)) for (est, true_r) in
user_ratings[:k])

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1
```

```
        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

    return precisions, recalls;
def get_precision_vs_recall(data_train, data_test, algo, vecK, verbose = False):
    precision_list = []
    recall_list = []
    f1_score_list = []
    if algo:
        for k_curr in vecK:
            algo.fit(data_train)
            predictions = algo.test(data_test)

            # Get precision and recall at k metrics for each user
            precisions, recalls = precision_recall_at_k(predictions, k = k_curr, threshold = 4)

            # Averaged over all users
            precision = sum(prec for prec in precisions.values()) / len(precisions)
            recall = sum(rec for rec in recalls.values()) / len(recalls)
            f1_score = 2 * (precision * recall) / (precision + recall)

            # Save measures
            precision_list.append(precision)
            recall_list.append(recall)
            f1_score_list.append(f1_score)

            if verbose:
                #print('K =', k_curr, '- Precision:', precision, ', Recall:', recall, ', F1 score:', f1_score)
                print("K = {}, Precision = {:.2f}, Recall = {:.2f}, F1 score = {:.2f}".format(k_curr,
precision, recall, f1_score))

    return {'precision': precision_list, 'recall': recall_list, 'f1_score': f1_score_list};
```

SVD

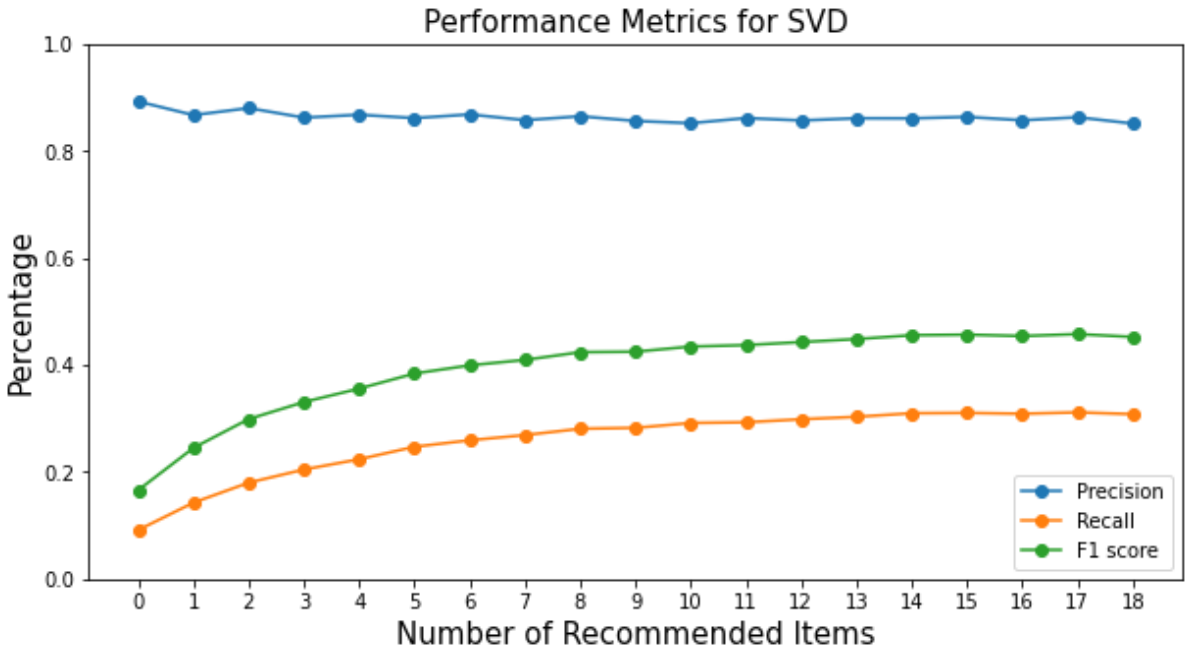
```
k_values = range(1,20)
metricsSVD = get_precision_vs_recall(trainset, testset, SVD(), k_values, True)
K = 1, Precision = 0.89, Recall = 0.09, F1 score = 0.17
K = 2, Precision = 0.87, Recall = 0.14, F1 score = 0.25
K = 3, Precision = 0.88, Recall = 0.18, F1 score = 0.30
K = 4, Precision = 0.86, Recall = 0.20, F1 score = 0.33
K = 5, Precision = 0.87, Recall = 0.22, F1 score = 0.36
K = 6, Precision = 0.86, Recall = 0.25, F1 score = 0.38
K = 7, Precision = 0.87, Recall = 0.26, F1 score = 0.40
K = 8, Precision = 0.86, Recall = 0.27, F1 score = 0.41
K = 9, Precision = 0.87, Recall = 0.28, F1 score = 0.42
K = 10, Precision = 0.86, Recall = 0.28, F1 score = 0.43
K = 11, Precision = 0.85, Recall = 0.29, F1 score = 0.43
K = 12, Precision = 0.86, Recall = 0.29, F1 score = 0.44
```

```
K = 13, Precision = 0.86, Recall = 0.30, F1 score = 0.44
K = 14, Precision = 0.86, Recall = 0.30, F1 score = 0.45
K = 15, Precision = 0.86, Recall = 0.31, F1 score = 0.46
K = 16, Precision = 0.86, Recall = 0.31, F1 score = 0.46
K = 17, Precision = 0.86, Recall = 0.31, F1 score = 0.45
K = 18, Precision = 0.86, Recall = 0.31, F1 score = 0.46
K = 19, Precision = 0.85, Recall = 0.31, F1 score = 0.45
```

```
# Get data
c1_SVD = metricsSVD['precision']
c2_SVD = metricsSVD['recall']
c3_SVD = metricsSVD['f1_score']
x_SVD = np.arange(len(c1_SVD))

# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_SVD), max(x_SVD) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_SVD, c1_SVD, marker = 'o')
ax1.plot(x_SVD, c2_SVD, marker = 'o')
ax1.plot(x_SVD, c3_SVD, marker = 'o')
#ax1.axvline(x = 10, color = "#8b0000", linestyle = "--")

# Chart setup
plt.title("Performance Metrics for SVD", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```



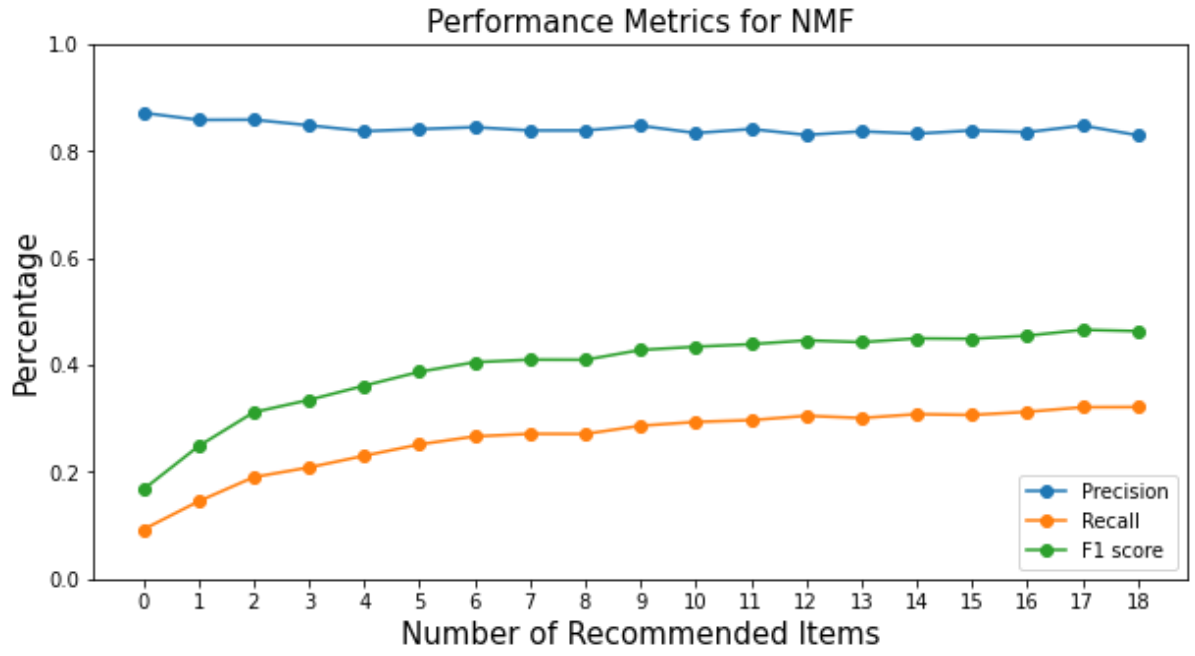
NMF

```
metricsNMF = get_precision_vs_recall(trainset, testset, NMF(), k_values, True)
K = 1, Precision = 0.87, Recall = 0.09, F1 score = 0.17
K = 2, Precision = 0.86, Recall = 0.15, F1 score = 0.25
K = 3, Precision = 0.86, Recall = 0.19, F1 score = 0.31
K = 4, Precision = 0.85, Recall = 0.21, F1 score = 0.33
K = 5, Precision = 0.84, Recall = 0.23, F1 score = 0.36
K = 6, Precision = 0.84, Recall = 0.25, F1 score = 0.39
K = 7, Precision = 0.84, Recall = 0.27, F1 score = 0.41
K = 8, Precision = 0.84, Recall = 0.27, F1 score = 0.41
K = 9, Precision = 0.84, Recall = 0.27, F1 score = 0.41
K = 10, Precision = 0.85, Recall = 0.29, F1 score = 0.43
K = 11, Precision = 0.83, Recall = 0.29, F1 score = 0.43
K = 12, Precision = 0.84, Recall = 0.30, F1 score = 0.44
K = 13, Precision = 0.83, Recall = 0.31, F1 score = 0.45
K = 14, Precision = 0.84, Recall = 0.30, F1 score = 0.44
K = 15, Precision = 0.83, Recall = 0.31, F1 score = 0.45
K = 16, Precision = 0.84, Recall = 0.31, F1 score = 0.45
K = 17, Precision = 0.84, Recall = 0.31, F1 score = 0.45
K = 18, Precision = 0.85, Recall = 0.32, F1 score = 0.47
K = 19, Precision = 0.83, Recall = 0.32, F1 score = 0.46

c1_NMF = metricsNMF['precision']
c2_NMF = metricsNMF['recall']
c3_NMF = metricsNMF['f1_score']
x_NMF = np.arange(len(c1_NMF))

# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_NMF), max(x_NMF) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_NMF, c1_NMF, marker = 'o')
ax1.plot(x_NMF, c2_NMF, marker = 'o')
ax1.plot(x_NMF, c3_NMF, marker = 'o')

plt.title("Performance Metrics for NMF", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```



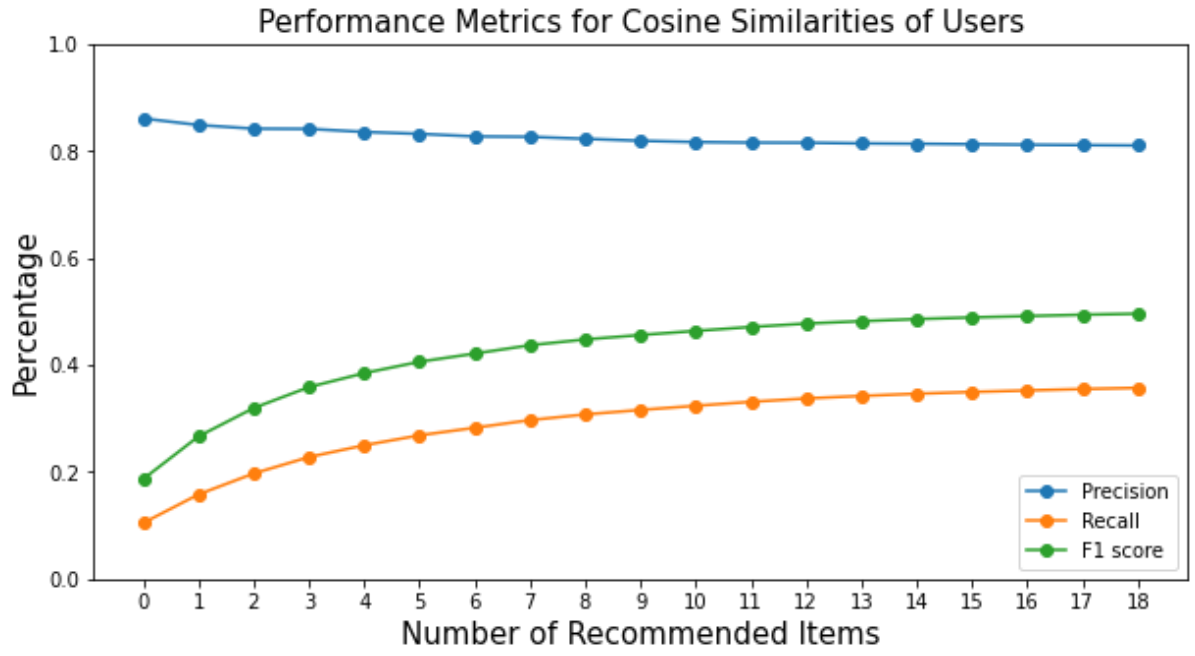
Cosine Similarity of the Users

metricsCosUser = get_precision_vs_recall(trainset, testset,algoCosUser, k_values, *True*)
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 1, Precision = 0.86, Recall = 0.10, F1 score = 0.19
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 2, Precision = 0.85, Recall = 0.16, F1 score = 0.27
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 3, Precision = 0.84, Recall = 0.20, F1 score = 0.32
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 4, Precision = 0.84, Recall = 0.23, F1 score = 0.36
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 5, Precision = 0.84, Recall = 0.25, F1 score = 0.38
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 6, Precision = 0.83, Recall = 0.27, F1 score = 0.41
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 7, Precision = 0.83, Recall = 0.28, F1 score = 0.42
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 8, Precision = 0.83, Recall = 0.30, F1 score = 0.44
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 9, Precision = 0.82, Recall = 0.31, F1 score = 0.45
Computing the cosine similarity matrix...
Done computing similarity matrix.

```
K = 10, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 11, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 12, Precision = 0.82, Recall = 0.33, F1 score = 0.47
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 13, Precision = 0.82, Recall = 0.34, F1 score = 0.48
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 14, Precision = 0.81, Recall = 0.34, F1 score = 0.48
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 15, Precision = 0.81, Recall = 0.35, F1 score = 0.49
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 16, Precision = 0.81, Recall = 0.35, F1 score = 0.49
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 17, Precision = 0.81, Recall = 0.35, F1 score = 0.49
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 18, Precision = 0.81, Recall = 0.36, F1 score = 0.49
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 19, Precision = 0.81, Recall = 0.36, F1 score = 0.50
c1_CosUser = metricsCosUser['precision']
c2_CosUser = metricsCosUser['recall']
c3_CosUser = metricsCosUser['f1_score']
x_CosUser = np.arange(len(c1_CosUser))

# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_CosUser), max(x_CosUser) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_CosUser, c1_CosUser, marker = 'o')
ax1.plot(x_CosUser, c2_CosUser, marker = 'o')
ax1.plot(x_CosUser, c3_CosUser, marker = 'o')

plt.title("Performance Metrics for Cosine Similarities of Users", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```

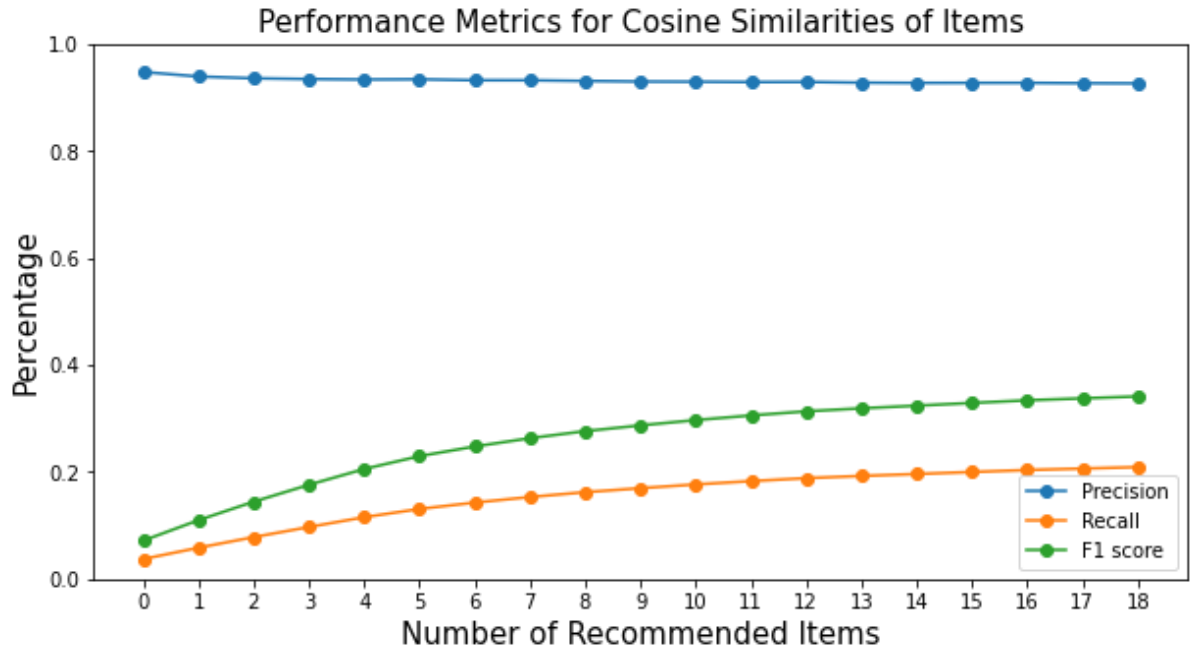



```
metricsCosItem = get_precision_vs_recall(trainset, testset,algoCosItem, k_values, True)
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 1, Precision = 0.95, Recall = 0.04, F1 score = 0.07
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 2, Precision = 0.94, Recall = 0.06, F1 score = 0.11
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 3, Precision = 0.94, Recall = 0.08, F1 score = 0.14
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 4, Precision = 0.93, Recall = 0.10, F1 score = 0.18
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 5, Precision = 0.93, Recall = 0.12, F1 score = 0.21
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 6, Precision = 0.93, Recall = 0.13, F1 score = 0.23
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 7, Precision = 0.93, Recall = 0.14, F1 score = 0.25
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 8, Precision = 0.93, Recall = 0.15, F1 score = 0.26
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 9, Precision = 0.93, Recall = 0.16, F1 score = 0.28
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 10, Precision = 0.93, Recall = 0.17, F1 score = 0.29
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 11, Precision = 0.93, Recall = 0.18, F1 score = 0.30
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 12, Precision = 0.93, Recall = 0.18, F1 score = 0.31
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 13, Precision = 0.93, Recall = 0.19, F1 score = 0.31
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 14, Precision = 0.93, Recall = 0.19, F1 score = 0.32
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 15, Precision = 0.93, Recall = 0.20, F1 score = 0.32
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 16, Precision = 0.93, Recall = 0.20, F1 score = 0.33
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 17, Precision = 0.93, Recall = 0.20, F1 score = 0.33
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 18, Precision = 0.93, Recall = 0.21, F1 score = 0.34
Computing the cosine similarity matrix...
Done computing similarity matrix.
K = 19, Precision = 0.93, Recall = 0.21, F1 score = 0.34
c1_CosItem = metricsCosItem['precision']
c2_CosItem = metricsCosItem['recall']
c3_CosItem = metricsCosItem['f1_score']
x_CosItem = np.arange(len(c1_CosItem))

# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_CosItem), max(x_CosItem) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_CosItem, c1_CosItem, marker = 'o')
ax1.plot(x_CosItem, c2_CosItem, marker = 'o')
ax1.plot(x_CosItem, c3_CosItem, marker = 'o')

plt.title("Performance Metrics for Cosine Similarities of Items", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```

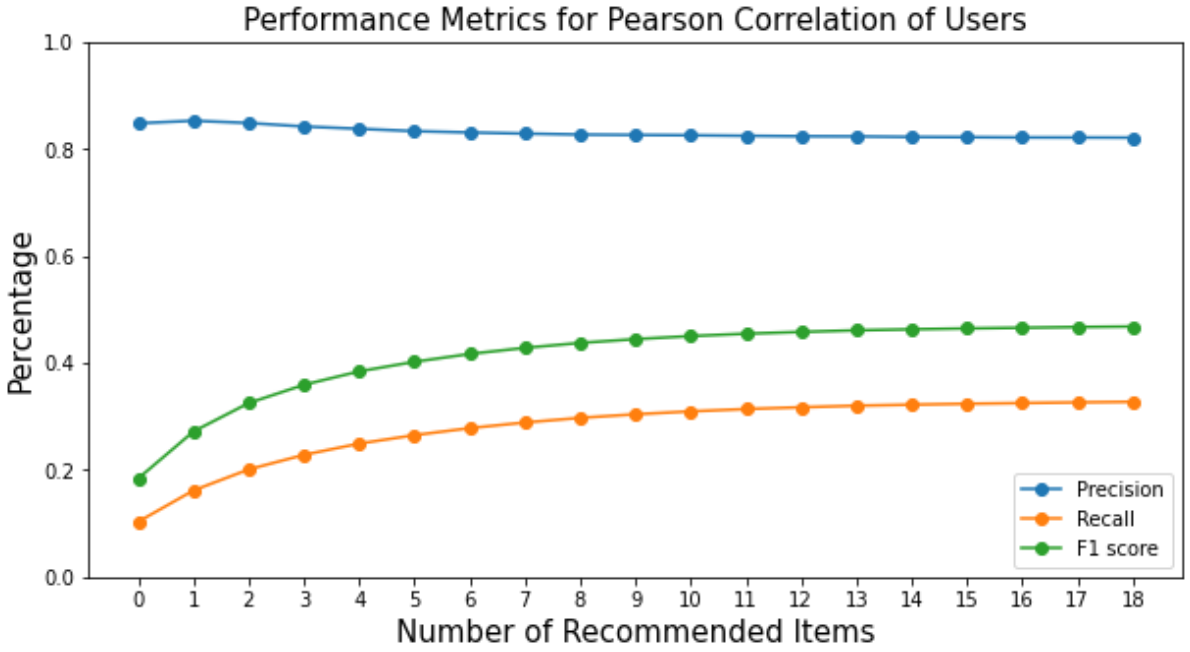



```
metricsPearUser = get_precision_vs_recall(trainset, testset,algoPearUser, k_values, True)
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 1, Precision = 0.85, Recall = 0.10, F1 score = 0.18
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 2, Precision = 0.85, Recall = 0.16, F1 score = 0.27
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 3, Precision = 0.85, Recall = 0.20, F1 score = 0.33
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 4, Precision = 0.84, Recall = 0.23, F1 score = 0.36
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 5, Precision = 0.84, Recall = 0.25, F1 score = 0.38
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 6, Precision = 0.83, Recall = 0.27, F1 score = 0.40
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 7, Precision = 0.83, Recall = 0.28, F1 score = 0.42
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

```
K = 8, Precision = 0.83, Recall = 0.29, F1 score = 0.43
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 9, Precision = 0.83, Recall = 0.30, F1 score = 0.44
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 10, Precision = 0.83, Recall = 0.30, F1 score = 0.44
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 11, Precision = 0.83, Recall = 0.31, F1 score = 0.45
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 12, Precision = 0.82, Recall = 0.31, F1 score = 0.45
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 13, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 14, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 15, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 16, Precision = 0.82, Recall = 0.32, F1 score = 0.46
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 17, Precision = 0.82, Recall = 0.33, F1 score = 0.47
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 18, Precision = 0.82, Recall = 0.33, F1 score = 0.47
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 19, Precision = 0.82, Recall = 0.33, F1 score = 0.47
c1_PearUser = metricsPearUser['precision']
c2_PearUser = metricsPearUser['recall']
c3_PearUser = metricsPearUser['f1_score']
x_PearUser = np.arange(len(c1_PearUser))
```

```
# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_PearUser), max(x_PearUser) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_PearUser, c1_PearUser, marker = 'o')
ax1.plot(x_PearUser, c2_PearUser, marker = 'o')
ax1.plot(x_PearUser, c3_PearUser, marker = 'o')

plt.title("Performance Metrics for Pearson Correlation of Users", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```




```
metricsPearItem = get_precision_vs_recall(trainset, testset, algoPearItem, k_values, True)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 1, Precision = 0.87, Recall = 0.07, F1 score = 0.13
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 2, Precision = 0.87, Recall = 0.12, F1 score = 0.21
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 3, Precision = 0.87, Recall = 0.15, F1 score = 0.26
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 4, Precision = 0.87, Recall = 0.18, F1 score = 0.30
Estimating biases using als...
```

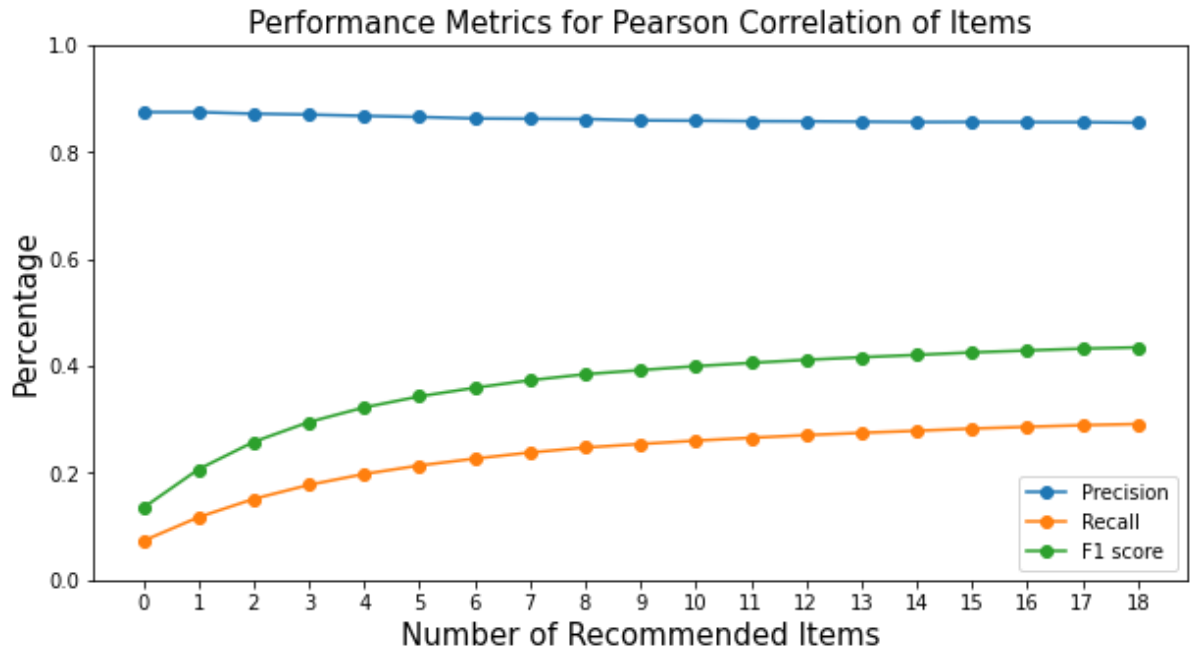
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 5, Precision = 0.87, Recall = 0.20, F1 score = 0.32
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 6, Precision = 0.87, Recall = 0.21, F1 score = 0.34
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 7, Precision = 0.86, Recall = 0.23, F1 score = 0.36
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 8, Precision = 0.86, Recall = 0.24, F1 score = 0.37
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 9, Precision = 0.86, Recall = 0.25, F1 score = 0.38
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 10, Precision = 0.86, Recall = 0.25, F1 score = 0.39
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 11, Precision = 0.86, Recall = 0.26, F1 score = 0.40
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 12, Precision = 0.86, Recall = 0.27, F1 score = 0.41
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 13, Precision = 0.86, Recall = 0.27, F1 score = 0.41
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 14, Precision = 0.86, Recall = 0.28, F1 score = 0.42
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 15, Precision = 0.86, Recall = 0.28, F1 score = 0.42
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 16, Precision = 0.86, Recall = 0.28, F1 score = 0.43
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.

```
K = 17, Precision = 0.86, Recall = 0.29, F1 score = 0.43
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 18, Precision = 0.86, Recall = 0.29, F1 score = 0.43
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
K = 19, Precision = 0.86, Recall = 0.29, F1 score = 0.43
```

```
c1_PearItem = metricsPearItem['precision']
c2_PearItem = metricsPearItem['recall']
c3_PearItem = metricsPearItem['f1_score']
x_PearItem = np.arange(len(c1_PearItem))

# Set up the matplotlib figure
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.xticks(np.arange(min(x_PearItem), max(x_PearItem) + 1, 1.0))
plt.ylim(0, 1)
ax1.plot(x_PearItem, c1_PearItem, marker = 'o')
ax1.plot(x_PearItem, c2_PearItem, marker = 'o')
ax1.plot(x_PearItem, c3_PearItem, marker = 'o')

plt.title("Performance Metrics for Pearson Correlation of Items", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("Precision", "Recall", "F1 score"), loc = "best")
plt.show()
```



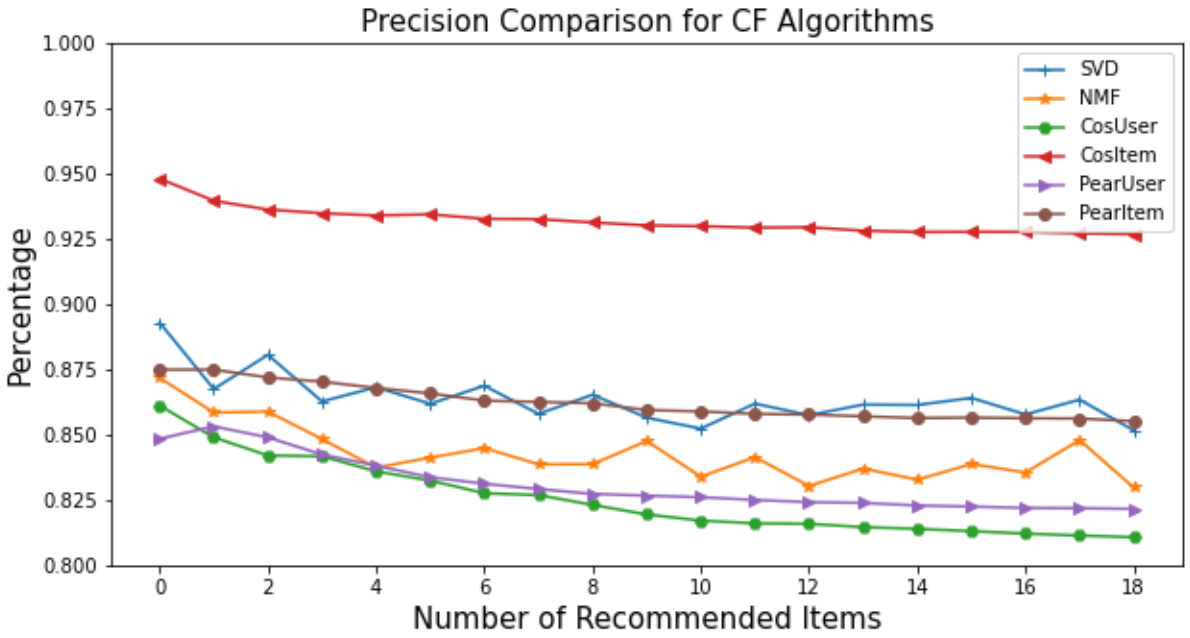
Comparing CF Algorithms

Precision

```
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.ylim(0.8, 1)
ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
ax1.plot(x_SVD, c1_SVD, marker = '+')
ax1.plot(x_NMF, c1_NMF, marker = '*')
ax1.plot(x_CosUser, c1_CosUser, marker = 'H')
ax1.plot(x_CosItem, c1_CosItem, marker = '<')
ax1.plot(x_PearUser, c1_PearUser, marker = '>')
ax1.plot(x_PearItem, c1_PearItem, marker = 'o')

plt.title("Precision Comparison for CF Algorithms", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("SVD", "NMF", "CosUser", "CosItem", "PearUser", "PearItem"), loc = "best")


plt.show()
```

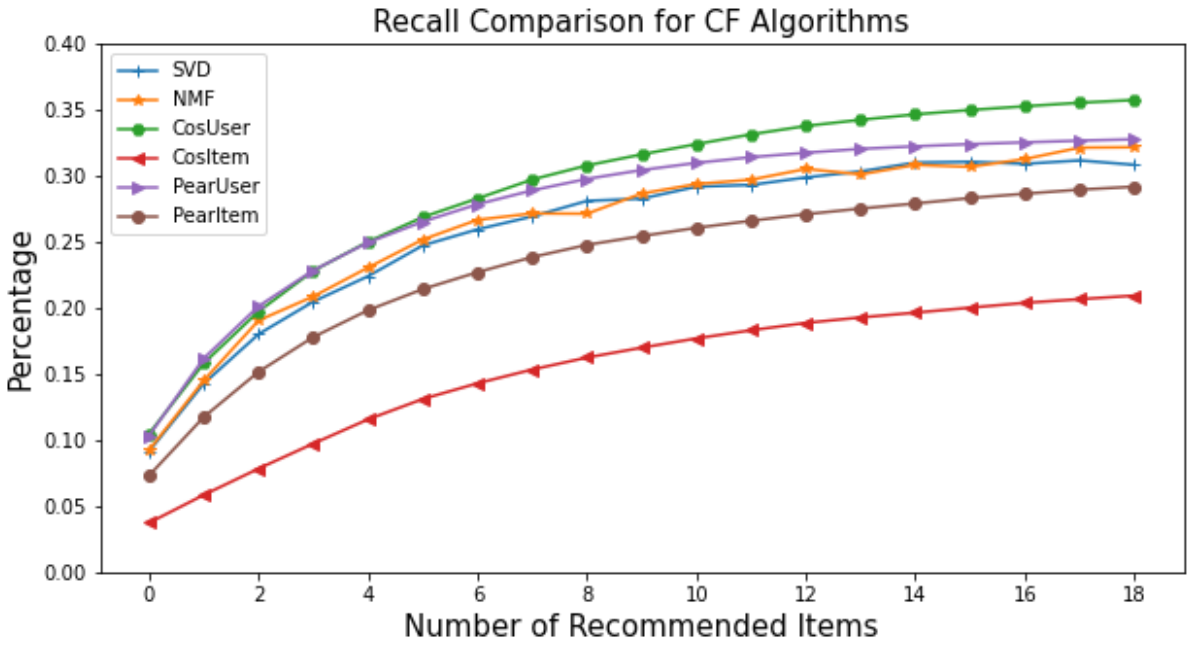


Recall

```
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.ylim(0, 0.4)
ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
ax1.plot(x_SVD, c2_SVD, marker = '+')
ax1.plot(x_NMF, c2_NMF, marker = '*')
ax1.plot(x_CosUser, c2_CosUser, marker = 'H')
ax1.plot(x_CosItem, c2_CosItem, marker = '<')
ax1.plot(x_PearUser, c2_PearUser, marker = '>')
ax1.plot(x_PearItem, c2_PearItem, marker = 'o')

plt.title("Recall Comparison for CF Algorithms", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
```

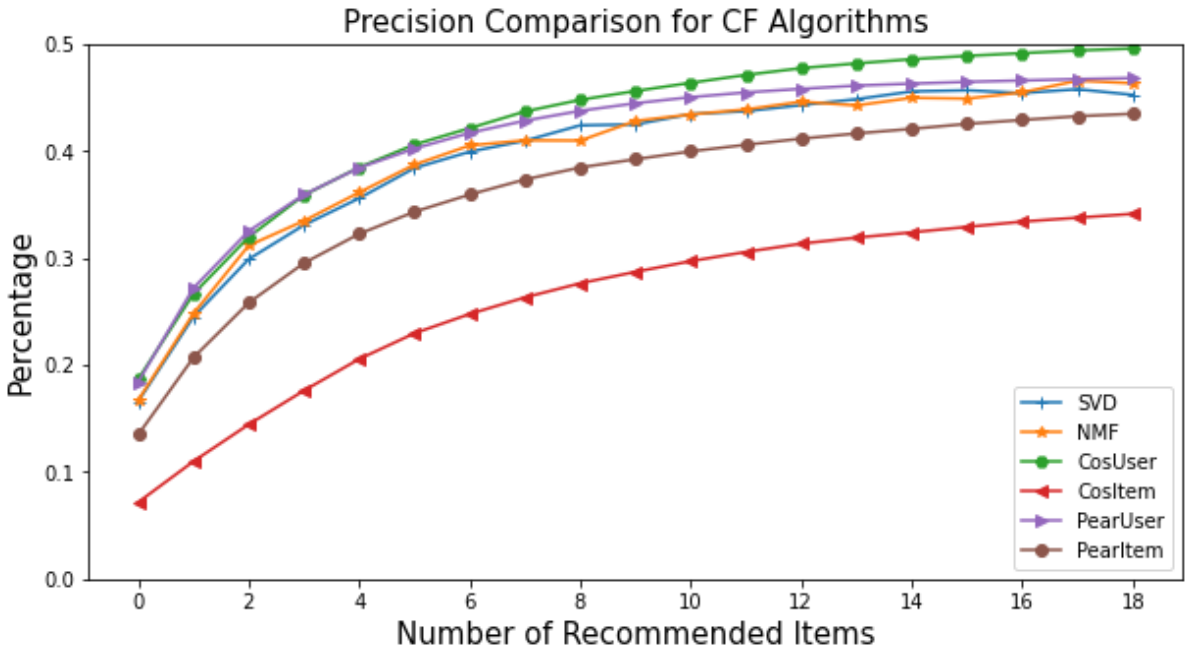
```
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("SVD", "NMF", "CosUser", "CosItem", "PearUser", "PearItem"), loc = "best")
plt.show()
```



F1-Score

```
fig, ax1 = plt.subplots(figsize = (10, 5))
plt.ylim(0, .5)
ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
ax1.plot(x_SVD, c3_SVD, marker = '+')
ax1.plot(x_NMF, c3_NMF, marker = '*')
ax1.plot(x_CosUser, c3_CosUser, marker = 'H')
ax1.plot(x_CosItem, c3_CosItem, marker = '<')
ax1.plot(x_PearUser, c3_PearUser, marker = '>')
ax1.plot(x_PearItem, c3_PearItem, marker = 'o')

plt.title("Precision Comparison for CF Algorithms", fontsize = 15)
plt.xlabel("Number of Recommended Items", fontsize = 15)
plt.ylabel("Percentage", fontsize = 15)
plt.legend(("SVD", "NMF", "CosUser", "CosItem", "PearUser", "PearItem"), loc = "best")
plt.show()
```


Precision Comparison for CF Algorithms
Percentage
Number of Recommended Items
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
12
14
16
18
SVD
NMF
CosUser
CosItem
PearUser
PearItem