Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering
Academic Year 2021-2022

Comparative study of Machine Learning algorithms for a book recommendation system

Machine Learning Laboratory

By

Junaid Girkar	**60004190057**
Kartik Suvarna	**60004190060**
Khushi Chavan	**60004190061**

Guide(s):

Prof. Khushali Deulkar

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering
Academic Year 2021-2022

Comparative study of Machine Learning algorithms for a book recommendation system

1. PROBLEM STATEMENT

This study provides an overview of recommendation systems and machine learning and their types. It outlines the types of machine learning such as supervised, unsupervised, semi-supervised learning, and reinforcement in brief. It explores how to implement recommendation systems using three types of filtering techniques: collaborative filtering, content-based filtering, and hybrid filtering. The machine learning techniques explained are clustering, co-clustering, and matrix factorization methods such as Single value decomposition (SVD) and Non-negative matrix factorization (NMF). It also discusses about different types of K-nearest neighbors (KNN). The evaluation of these algorithms is performed on the basis of three metric parameters: F1 measurement, Root mean squared error (RMSE) and Mean absolute error (MAE). For the experimentation, this study uses the BookCrossing dataset and compares analysis based on metric parameters.

INTRODUCTION

a. Need

Recommendation systems are widely used today to recommend products to users based on their interests. A recommendation system is one of the strongest systems for increasing profits by retaining more users in a very big competition. Online book reading and selling websites like Kindle and Goodreads compete against each other on many factors. One of those important factors is their book recommendation system. A book recommendation system is designed to recommend books of interest to the buyer.

The purpose of a book recommendation system is to predict buyer's interest and recommend books to them accordingly. A book recommendation system can take into account many parameters like book content and book quality by filtering user reviews.

b. Working

Dataset

The BookCrossing dataset is built by CAI-Nicolas Ziegler from Amazon Web Services. There are 270,000 books read by 90,000 users with 1.1 million reviews. The data consist of three tables which include information about ratings, books, and users. This data is downloaded from Kaggle. The rating dataset provides a list of book ratings given by the users. It includes 1,149,780 rating records containing 3 fields: userID, ISBN, and bookRating. The ratings are either explicitly expressed on a scale of 1 to 10 or implicitly expressed by zero. As shown in Fig. 1, the vast majority of ratings are 0 and these ratings are distributed very unevenly. The books dataset provides book information, which includes 271,360 book records containing 8 fields. First, 5 fields containing the content-based information: ISBN, Book-Title, Book-Author, Year-Of-Publication, Publisher, and the last 3 image-URL fields: Image-URL-S, Image-URL- M, Image-URL-L. These 3 different URL images are linked to the cover page of the books according to their size. The user dataset provides demographic information of users. It includes 278,858 user records and 3 fields: user id, Location, and Age. Fig.2 shows that the majority of active users are youth between the ages of 20 and 30.

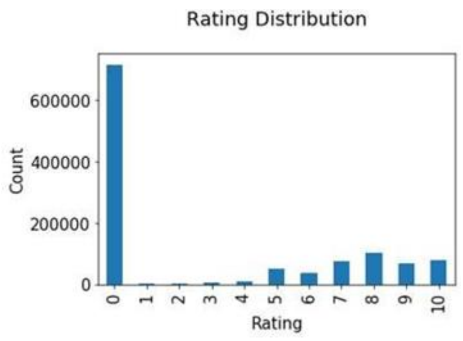


Figure 1: Rating Distribution of the books in dataset

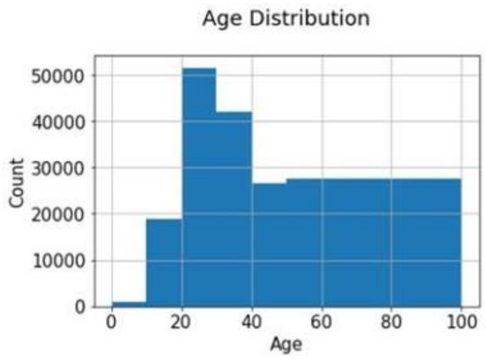


Figure 2: Age Distribution of users in user-data

Libraries used

NumPy

NumPy is a fundamental package and linear algebra library that is an essential tool for scientific computations & mathematical operations in python. It is a key component of research analysis pipelines in a wide range of fields such as physics, chemistry, astronomy, geology, biology, psychology, materials science, engineering, finance, and economics. For example, in astronomy, NumPy was an integral part of the discovery of gravitational waves as well as the first imaging of a black hole.

NumPy is also an open-source, community-developed library, which provides multidimensional arrays along with array-aware functions. Since NumPy array is simple and easy to use, it is the de-facto format for displaying array data in Python.

Surprise

Surprise is an easy-to-use Python scikit for recommender systems. It utilizes built-in Python data structures (primarily dictionaries) and NumPy arrays. It provides a collection of estimation algorithms for rating prediction. It also includes the implementation of classical algorithms such as the main similarity-based algorithms, and matrix factorization algorithms like SVD or NMF. Moreover, it supports tools for model evaluation, including cross-validation iterators, built-in metrics, and tools for selecting models and searching for hyperparameters, such as grid search and randomized search. Lastly, it was designed to be helpful to researchers who want to quickly explore new recommendation ideas through the creation of custom prediction algorithms.

Scikit-learn

Scikit-learn is a Python module that integrates a wide variety of ultra-modern computer learning algorithms for solving medium-scale supervised and unsupervised problems. This bundle aims to make the knowledge of machine learning easily accessible to non-specialists using a general-purpose high-level language. The scikit-learn library provides several efficient tools for machine learning and statistical modeling such as classification, regression, clustering, and dimensionality reduction. It is used to build machine learning models instead of reading the data, manipulating, and summarizing it.

Implementation

The book recommendation system has been done using item-based and user-based collaborative filtering experimented in python and compiled in Jupyter Notebook. After evaluating the RMSE scores of the user and the item, optimization of the book recommendation system is done by integrating various other algorithms such as co- clustering, SVD, NMF, KNNbasic, KNNwithMeans, and KNNwithZScore models from the Surprise library.

c. Applications

Over the last few years, the enthusiasm for recommendation systems has increased tremendously. This is the most widely used service on high-end websites like Amazon, Google, YouTube, Netflix, IMDb, TripAdvisor, Kindle, etc. A number of media companies develop these systems as a service model to their clients. Furthermore, the implementation of such systems at commercial and non-profit sites attracts the attention of the customer. These also satisfy clients more with online research results. These systems help customers search for their loved items faster and acquire more authentic predictions leading to higher sales at an eCommerce site.

2. ALGORITHMS USED

Co-clustering

Co-clustering, also known as bi-clustering, is a method wherein there is a simultaneous clustering between rows and columns of a matrix. This matrix represents information as a function of user characteristics and item characteristics. In other words, co-clustering can also be visualized as grouping two different kinds of entities according to their similarity. The result of a co-clustering algorithm is commonly termed a bi-cluster. The kinds of bi-clustering are classified according to the nature of these bi-clusters. It depends mainly upon constant and consistent values.

1) Bi-cluster with constant values: Rows and columns within a clustering block have the same constant value.
2) Bi-cluster with constant values in rows or columns: Every row or column in a clustering block has the same constant value.
3) Bi-cluster with coherent values: These bi-clusters identify more complex similarities between genes and conditions using an additive or multiplicative method.

The clustering technique is implemented through a variety of matrix factorization techniques.

Singular Value Decomposition

This method is associated with linear algebra and is increasingly popular within ML algorithms. Its application is mainly recommendation systems for e-commerce, music, or video streaming sites.

SVD refers to the decomposition of a single matrix into three additional matrices. The general form is: $M = XSY$T

where M is the given mxn matrix,

X is an mxn orthogonal matrix that denotes the relation between the user and latent factors,

S is an nxn diagonal matrix that denotes the strength of these latent factors, and

Y is nxn orthogonal matrix and it represents the similarity between the user and latent factors.

The steps involved in SVD are given below.

1. In the first step, the data is represented as a matrix with rows as user and columns as items.
2. If there are any empty entries in the matrix, provide the average of the other entries so that there is no major error in the calculation.
3. After this, compute the SVD. (Done using NumPy and Surprise library)
4. After calculating the SVD, you only need to reduce it to obtain the expected matrix that will be used for the prediction by looking at the appropriate user/article pair.

The primary benefit of SVD is that it simplifies the data set and eliminates noise from the data set. It also functions with the numerical data set. Also, it could improve the precision. There are many issues related to the SVD. One of the most important issue is data scarcity, also called the cold start problem. This occurs due to a new community, user, or item. If a new community, user, or item is added, the recommendation system will not work properly due to a lack of information. Black sheep is also an issue, meaning some customers also agree and disagree with the same group of people. If so, it is impossible to make recommendations. Due to its temporal complexity (O (n)), it also suffers from scalability issues.

There are different applications of SVD. The most common applications are pseudo-inverse, resolving homogeneous linear equations, minimizing total least squares, range, null space and rank, and approximation to the lowest rank matrix. In addition, it is used for signal processing, image processing, and big data.

Non-negative Matrix Factorization

This is also a matrix factorization technique. As with SVD, the analogy for this approach is to break down or factorize a given matrix. The only difference, on the other hand, is that the matrix is split into two parts. The two parts are called W and H. W matrix is for weights which represent each column as a basic element. These are building blocks from which to obtain predictions to the original data item. H matrix is hidden which represents the coordinates of the data items of W. In other words, it guides us in converting to the original data item from the group of building blocks of W.

The order of execution in NMF is given below.

1. Import the NMF model using the Surprise library.
2. Then, load the dataset and isolate it to the given model.
3. Later, clean the data and create a function to pre-process data.

4. Successively create a document term matrix 'V'(given matrix).
5. Create a function to display the mode features.
6. Then, run NMF on the document term matrix 'V'.
7. Continue checking and iterating until useful features are found.

The advantage of NMF is that it breaks down the given matrix into two smaller matrices whose dimensions can be controlled by the given matrix. It differs from other matrix factorization algorithms because it works only on positive numbers which makes the data interpretable. The dataset can become smaller if W and H are depicted sparsely. The issue with the semi-supervised NMF is that depending on the number of data points available, there is a reduction in the fitted data points.

Applications of the NMF include the processing of audio spectrograms, document clustering, recommendation systems, chemometry, and many others. It is also used for dimensionality reduction in astronomy, statistical data imputation, as well as nuclear imaging.

Difference between SVD and NMF

So as stated above, both SVD and NMF are matrix factorization techniques. But there are also some differences between them, which could help us to choose the best algorithm for a situation between these two.

1. The SVD includes both negative and positive values, while the NMF has strictly positive values. That makes NMF useful because it provides more sense and connections are made easier.
2. SVD factors can be related to the eigen functions of a system where the original matrix denotes a system about which one is taking interest from a signal processing perspective. This makes SVD more effortless. Although NMF can also be used for the same purpose because the association is indirect in this approach, it becomes more tedious.
3. The factors of SVD are unique whereas the factors of NMF are not unique. As a result, NMF is better for algorithms with privacy protection.
4. SVD factors into three matrices out of which the sigma matrix gives the information stored in the vector. Whereas NMF only factors into two matrices which do not include the sigma matrix.

K-Nearest Neighbors

KNN is an easy machine learning algorithm based on supervised ML learning. It finds similar items based on the distance between test data and individual training data using a variety of distance concepts. In this algorithm, predictions are mainly made using the calculation of the Euclidean distance of the nearest neighbors. Besides, the use of Jaccard similarity, Minkowski, Manhattan, or Hamming distance can be done instead of Euclidean. This is a non-parametric algorithm that assumes nothing on the given data. It is also referred to as a lazy learning algorithm, which does not learn from data, instead stores and performs actions on the data.

The steps involved in KNN are given below.

1. Load the dataset and preprocess it.
2. Fit the KNN algorithm (defined as Nearest-Neighbors) to the training dataset (use the scikit-learn library). For using the Surprise library, it is defined as KNNBasic.
3. Predict the test result.
4. Creating the confusion matrix and finding the test accuracy of the result.
5. After this, the visualization of the test result can be done.

This algorithm is used as it is easy to interpret the result. It also has great predictive power and less computing time. The main issue with KNN is that it becomes much slower as the volume of data increases. As such, it does not give good accuracy with large datasets. It is also highly sensitive to missing values, outliers, and noise from the dataset.

It is primarily used for classification and regression problems. The result of a classification problem is a discrete value while for a regression problem, the result is a real number (containing a decimal). It is commonly used for text extraction. It is used in finance for stock prediction, management of loans, and analysis of money laundering. It is used in agriculture for weather forecasting and estimation of soil water parameters. It is also used in medicine to predict different diseases.

Evaluation Methods

There are various methods used in the evaluation of machine learning methods. One of the commonly used methods is the absolute error and accuracy-based evaluation methods such as RMSE (Root mean squared Error), MSE (Mean square error), and MAE (Mean absolute error). There are decision support methods like precision, recall, F1-measure, and ROC (Receiver operating characteristic) curve. In addition, there are ranking-based evaluation methods, such as nDCG (Normalization of discounted cumulative gain), MRR (Mean reciprocal rank), mean precision, and Spearman rank correlation. Moreover, different metric evaluation methods assess performance based on prediction, decision, and ranking power. Examples of these metric-based approaches include coverage, popularity, novelty, diversity, and temporal evaluation. Finally, business sector metrics can be used to reach its objective. The above- mentioned algorithms will be evaluated using F1-measure, RMSE, and MAE.

F1 measure

This accuracy measurement combines accuracy and recall and is also called the harmonic average of the model. This is used to measure the accuracy of the model.

The formula for the F1 measure is F1=2*P*R/(P+R), where P and R are the precision and recall of the model.

Precision: This measure, also known as the TP (True positives), is defined as the ratio of TP to the sum of TP and FP (False positives).

Recall: This measure, also known as sensitivity, is defined as the ratio of the TP to the sum of TP and FN (False negatives).

$$Precision = \frac{|TP|}{|TP + FP|} = \frac{|Recommended\ items\ that\ are\ suitable|}{|Recommended\ Items|}$$

$$Recall = \frac{|TP|}{|TP + FN|} = \frac{|Recommended\ items\ that\ are\ suitable|}{|Suitable\ Items|}$$

To avoid the least robustness of normal accuracy measurements, this measurement is preferred since it can take note of variations of different types of errors. The F1 measure is efficient whenever there is a presence of different costs of FP(False positives) and FN(False negatives). The F1 measurement can also be useful if there is an imbalance in the class feature numbers because in such cases the precision can be very misleading. The weakness of the F1 measurement is that the value calculated for one feature is independent of the other. In other words, it cannot compute the effectiveness of two features combined or based on each other's information. The applications for the F1 measurement include information retrieval in NLP (Natural Language Processing). This is most frequently used in search engine systems. In addition, it is most commonly used in binary classification systems.

RMSE (Root Mean Squared Error)

It is a performance measure of the ML models that are primarily calculated to see how well the model fits (i.e., less error, more accuracy). In other words, this is used to predict quantitative data. It is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

In the above RMSE equation, y_j is the original data and $\hat{y}_j$ is the predicted data.

This measure is used because it is quite easy to distinguish. This makes it easier to work with methods such as gradient descent. This is also good for evaluating the standard deviation for distributing the errors generated.

RMSE has square errors, so even a small error can affect the value immensely which allows us to ensure that the model yields as little error as possible. This means that an error of 10 will become 100 times worse than an error of 1. RMSE could become difficult to understand from an interpretation point of view as it contains square values whereas MAE would be clear to understand due to absolute values.

MAE (Mean Absolute Error)

This measure is also used as an alternative to RMSE. MAE is the average of the absolute difference between the original data and the predicted data. If this absolute value is not taken, this would become the mean bias error (MBE).

To represent MAE mathematically:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

In the above MAE equation, y_j is the original data and $\hat{y}_j$ is the predicted data. MAE is more stable than RMSE when the variation in frequency error distribution increases. This means that an error of 10 will be 10 times worse than an error of 1. MAE is generally preferable when scales of error are linear whereas RMSE is preferable when scales of error are non-linear. MAE is not useful when no absolute value is required, in such cases RMSE is preferable.

MAE	RMSE
It doesn't consider the sign of the input, if the input is negative it takes the positive value.	It considers the sign of the input whether it is positive or negative.
It is less biased towards large values. Thus, when it comes to a large error, it does not reflect the result of the algorithm.	When it comes to large errors, it reflects in the result of the algorithm. Thus, it is much better than MAE.
The MAE value is comparatively smaller as the sample size increases.	RMSE is comparatively higher than MAE for increasing sample size.
MAE restricts larger errors.	RMSE does not restrict large errors.
MAE is preferred where there is a proportion between overall performance and an increase in error.	RMSE is preferred where the overall performance and the increase in error are disproportionate.

3. IMPLEMENTATION

a. Code of important functions

Code for cleaning of data:

```
#import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from surprise import Reader, Dataset
from surprise import model_selection, accuracy
from surprise import NMF
from surprise import SVD
from surprise import CoClustering
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
plt.style.use('seaborn-white') # Use seaborn-style plots
%matplotlib inline

#Read files.
import os
global directory; directory = '../BookRating'
def files(): return os.listdir(directory)

Users = pd.read_csv('C:/Users/Khushi Nitin Chavan/OneDrive/Desktop/BX-CSV-
Dump/BX-Users.csv', sep = ";", encoding='cp1252')
Ratings = pd.read_csv('C:/Users/Khushi Nitin Chavan/OneDrive/Desktop/BX-
CSV-Dump/BX-Book-Ratings.csv', sep = ";", encoding='cp1252')
Books = pd.read_csv('C:/Users/Khushi Nitin Chavan/OneDrive/Desktop/BX-
CSV-Dump/BX-Books.csv', sep=";", usecols=range(5), encoding ='iso-8859-1')


#Clean data with replacement of missing and invalid data
#Users
Users.columns = Users.columns.str.strip().str.lower().str.replace('-', '_')
Users.user_id = Users.user_id.astype(int)
#User ages
#invalid ages of users such as, 224 years old and 0 years old observed
std = Users.age.std(); mode = int(Users.age.mode().values.tolist()[0]); len_fillna =
 len(Users.index)
#learn stasticis of data
df_copy = Users.copy()
df_normalize = df_copy.age.value_counts().rename_axis('age').reset_index(name
= 'user_counts')
df_normalize = df_normalize.sort_values(by=['age'])
df_normalize['id'] = df_copy['user_id']
```

```
df_normalize = df_normalize.assign(precent_counts=lambda x: x.user_counts.cu
msum() / x.user_counts.sum())
#define where the 0.5% to 99.5% user count of age interval is
age_high = max(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
(df_normalize['precent_counts'] <= 0.995)].age.unique().tolist())
age_low = min(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) & \
(df_normalize['precent_counts'] <= 0.995)].age.unique().tolist())
#fill invalid and missing ages with random normal distributed numbers
#shifting mean to mode(mean = mode in normaldistribution)
fillna_list = (np.random.normal(mode, std, 1000)).tolist()
Users.loc[(Users.age<5) | (Users.age>90), 'age'] = np.nan
Users.age = Users.age.fillna(pd.Series(fillna_list))
#User locations
#originally format: location : 'farnborough, hants, united kingdom'
#break into three columns to better check validity
tmp_list = Users.location.str.split(',', 2, expand=True)
tmp_list.describe(include=[object])
#fill nan with 'other'
tmp_list.fillna('other', inplace=True)
Users['city'] = tmp_list[0];Users['state'] = tmp_list[1];Users['country'] = tmp_list[2
]
Users.drop(columns=['location'], inplace=True)

#Books
Books.columns = Books.columns.str.strip().str.lower().str.replace('-', '_')
Books.year_of_publication = pd.to_numeric(Books.year_of_publication, errors='c
oerce')
#replace outscaled publication years
std = Books.year_of_publication.std()
mode = int(Books.year_of_publication.mode().values.tolist()[0])
len_fillna = len(Books.index)
#learn statistical information of book counts on year of publication
df_normalize={}
df_copy = Books.copy()
df_copy.year_of_publication.replace(0, np.nan, inplace=True)
df_normalize = df_copy.year_of_publication.value_counts().rename_axis('year').r
eset_index(name = 'book_counts')
df_normalize = df_normalize.sort_values(by=['year'])
df_normalize['id'] = df_copy['isbn']
df_normalize = df_normalize.assign(precent_counts=lambda x: x.book_counts.cu
msum() / x.book_counts.sum())
year_high = max(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
(df_normalize['precent_counts'] <= 0.995)].year.unique().tolist())
year_low = min(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
(df_normalize['precent_counts'] <= 0.995)].year.unique().tolist())
#fill outscaled year and missing year(originally nan, became 0 after to_numeric)
fillna_list = (np.random.normal(mode, std, 1000)).tolist()
```

```
Books.loc[(Books.year_of_publication<1960) | (Books.year_of_publication>2020
), 'year_of_publication'] = np.nan
Books.year_of_publication = Books.year_of_publication.fillna(pd.Series(fillna_lis
t))
#we cannot recommend books without a title...
Books.dropna(subset=['book_title'], inplace=True)

#Ratings
Ratings.columns = Ratings.columns.str.strip().str.lower().str.replace('-', '_')
#extract explict ratings: 1-10
Ratings_ex = Ratings[Ratings.book_rating != 0]
#rescale with corrected user and book IDs from Users and Books
Book_Ratings_ex = Ratings_ex[Ratings_ex.isbn.isin(Books.isbn)]
User_Book_Ratings_ex = Book_Ratings_ex[Book_Ratings_ex.user_id.isin(Users.
user_id)]
#Copy unprocessed df for plotting
User_Book_Ratings_ = User_Book_Ratings_ex.copy()#this only for plotting
#locate lazy users with ratings <10
df_copy = User_Book_Ratings_ex.copy()
df_normalize = df_copy.user_id.value_counts().rename_axis('user_id').reset_inde
x(name = 'ratings_counts')
df_normalize = df_normalize.sort_values(by=['ratings_counts'])
lazy_users = df_normalize.loc[(df_normalize.ratings_counts < 10),'user_id'].uniqu
e().tolist()
#remove rows with lazy users
User_Book_Ratings_ex.loc[(User_Book_Ratings_ex['user_id'].isin(lazy_users))]
= np.nan
User_Book_Ratings_ex.dropna(inplace=True)
#books with ratings
Book_with_r = User_Book_Ratings_ex.join(Books.set_index('isbn'), on='isbn')
#book,users with ratings
User_Book_r = Book_with_r.join(Users.set_index('user_id'), on='user_id')

#Surprise Read
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_ratin
g']], reader)
```

Code for calculating the precision, recall and f1-score:

```
 from collections import defaultdict
 def precision_recall_at_k(predictions, k = 10, threshold = 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 (es
t, 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(algo, k_max = 10, verbose = False):
    precision_list = []
    recall_list = []
    f1_score_list = []

    if algo:
        for k_curr in range(1, k_max + 1):
            algo.fit(data_train)
            predictions = algo.test(data_testset)

            # Get precision and recall at k metrics for each user
            precisions, recalls = precision_recall_at_k(predictions, k = k_curr, thresho
ld = 5)

            # Precision and recall can then be 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)

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

Code for SVD algorithm:

```
# Load SVD algorithm
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_ratin
g']], reader)
model = SVD()
# Train on books dataset
%time model_selection.cross_validate(model, data, measures=['RMSE','MAE'],c
v=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(model, k_max, True)
np.mean(metrics['f1_score'])
```

Code for NMF algorithm:

```
# Load NMF algorithm
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_ratin
g']], reader)
model1 = NMF()
# Train on books dataset
%time model_selection.cross_validate(model1, data, measures=['RMSE','MAE'],
cv=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(model1, k_max, True)
np.mean(metrics['f1_score'])
```

Code for Co-clustering algorithm:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_ratin
g']], reader)
CoClusteringmodel = CoClustering()
%time model_selection.cross_validate(CoClusteringmodel, data, measures=['RM
SE','MAE'], cv=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(CoClusteringmodel, k_max, True)
np.mean(metrics['f1_score'])
```

Code for KNNBasic algorithm:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_ratin
g']], reader)
KNNBasicmodel = KNNBasic()
%time model_selection.cross_validate(KNNBasicmodel, data, measures=['RMSE
','MAE'], cv=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(KNNBasicmodel, k_max, True)
np.mean(metrics['f1_score'])
```

Code for KNNWithMeans algorithm:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reader)
KNNWithMeansmodel = KNNWithMeans()
%time model_selection.cross_validate(KNNWithMeansmodel, data, measures=['RMSE','MAE'], cv=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(KNNWithMeansmodel, k_max, True)
np.mean(metrics['f1_score'])
```

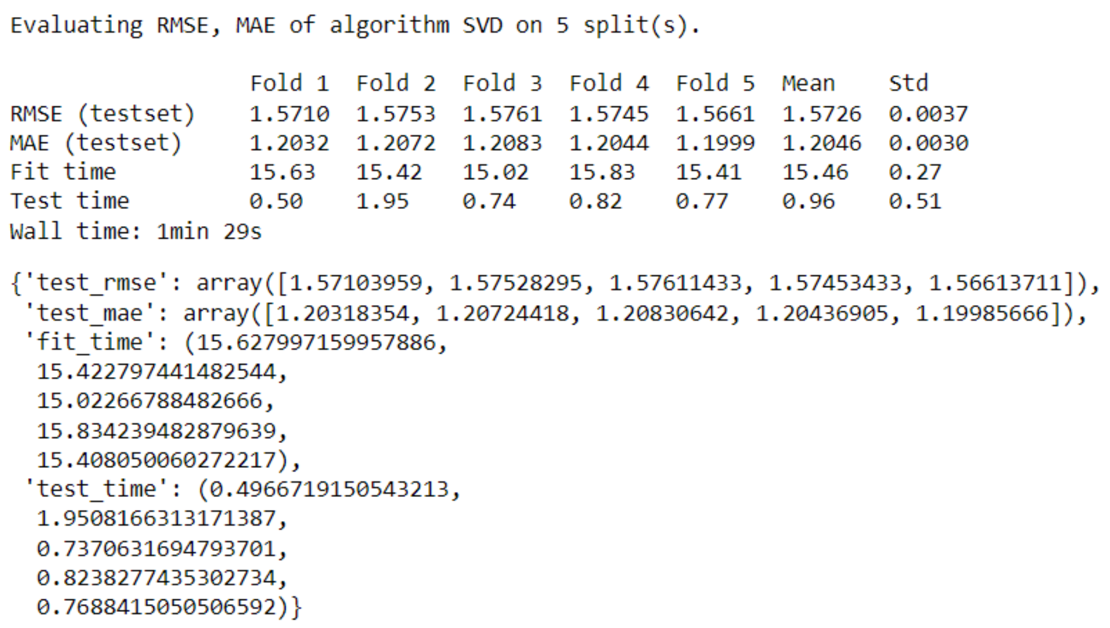
Code for KNNWithZscore algorithm:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reader)
KNNWithZScoremodel = KNNWithZScore()
%time model_selection.cross_validate(KNNWithZScoremodel, data, measures=['RMSE','MAE'], cv=5, verbose=True)
k_max = 10
metrics = get_precision_vs_recall(KNNWithZScoremodel, k_max, True)
np.mean(metrics['f1_score'])
```

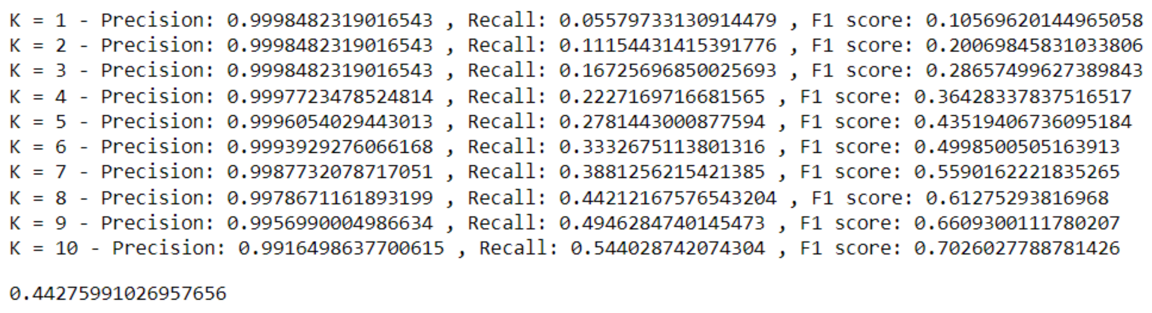
Code for comparison of RMSE, MAE AND F1-score:

```
import matplotlib.pyplot as plt
Algorithms = ['Co-clustering' ,'SVD','NMF','KNNBasic','KNNwithMeans','KNNwithZScore']
#mannullt recode...since re-running takes too much time
mae_results =[1.4286,1.2041,2.0782,1.5268,1.3931,1.3801]
rmse_results =[1.8393,1.5726,2.4767,1.9473,1.7994,1.7967]
f1_results =[0.4289,0.4428,0.4202,0.4434,0.4404,0.4402]
plt.figure(figsize=(18,7))
plt.title('Comparison of Algorithms on RMSE, MAE and F1-score', loc='center', fontsize=20)
plt.plot(Algorithms, rmse_results, label='RMSE', marker='o')
plt.plot(Algorithms, mae_results, label='MAE', marker='o')
plt.plot(Algorithms, f1_results, label='F1-measure', marker='o')
plt.xlabel('Algorithms', fontsize=16)
plt.ylabel('Error', fontsize=16)
plt.grid(ls='dashed')
plt.legend()
plt.show()
```

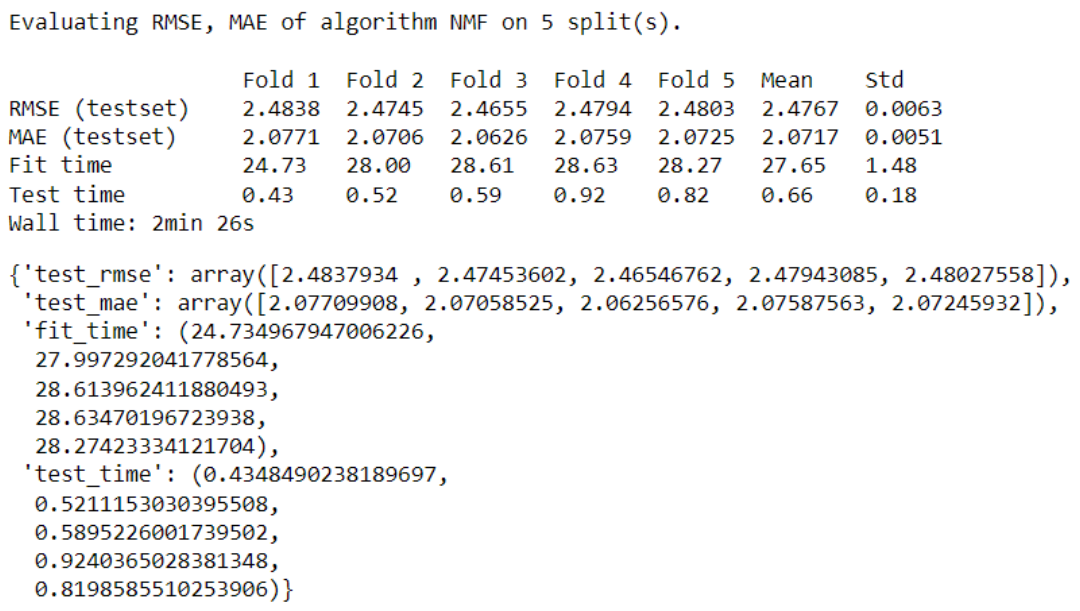
b. Screenshots



```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5710  1.5753  1.5761  1.5745  1.5661  1.5726  0.0037
MAE (testset)     1.2032  1.2072  1.2083  1.2044  1.1999  1.2046  0.0030
Fit time          15.63   15.42   15.02   15.83   15.41   15.46   0.27
Test time         0.50    1.95    0.74    0.82    0.77    0.96    0.51
Wall time: 1min 29s

{'test_rmse': array([1.57103959, 1.57528295, 1.57611433, 1.57453433, 1.56613711]),
 'test_mae': array([1.20318354, 1.20724418, 1.20830642, 1.20436905, 1.19985666]),
 'fit_time': (15.627997159957886,
  15.422797441482544,
  15.02266788482666,
  15.834239482879639,
  15.408050060272217),
 'test_time': (0.4966719150543213,
  1.9508166313171387,
  0.7370631694793701,
  0.8238277435302734,
  0.7688415050506592)}
```



```
K = 1 - Precision: 0.9998482319016543 , Recall: 0.05579733130914479 , F1 score: 0.10569620144965058
K = 2 - Precision: 0.9998482319016543 , Recall: 0.11154431415391776 , F1 score: 0.20069845831033806
K = 3 - Precision: 0.9998482319016543 , Recall: 0.16725696850025693 , F1 score: 0.28657499627389843
K = 4 - Precision: 0.9997723478524814 , Recall: 0.2227169716681565 , F1 score: 0.36428337837516517
K = 5 - Precision: 0.9996054029443013 , Recall: 0.2781443000877594 , F1 score: 0.43519406736095184
K = 6 - Precision: 0.9993929276066168 , Recall: 0.3332675113801316 , F1 score: 0.4998500505163913
K = 7 - Precision: 0.9987732078717051 , Recall: 0.3881256215421385 , F1 score: 0.5590162221835265
K = 8 - Precision: 0.9978671161893199 , Recall: 0.44212167576543204 , F1 score: 0.61275293816968
K = 9 - Precision: 0.9956990004986634 , Recall: 0.4946284740145473 , F1 score: 0.6609300111780207
K = 10 - Precision: 0.9916498637700615 , Recall: 0.544028742074304 , F1 score: 0.7026027788781426

0.44275991026957656
```

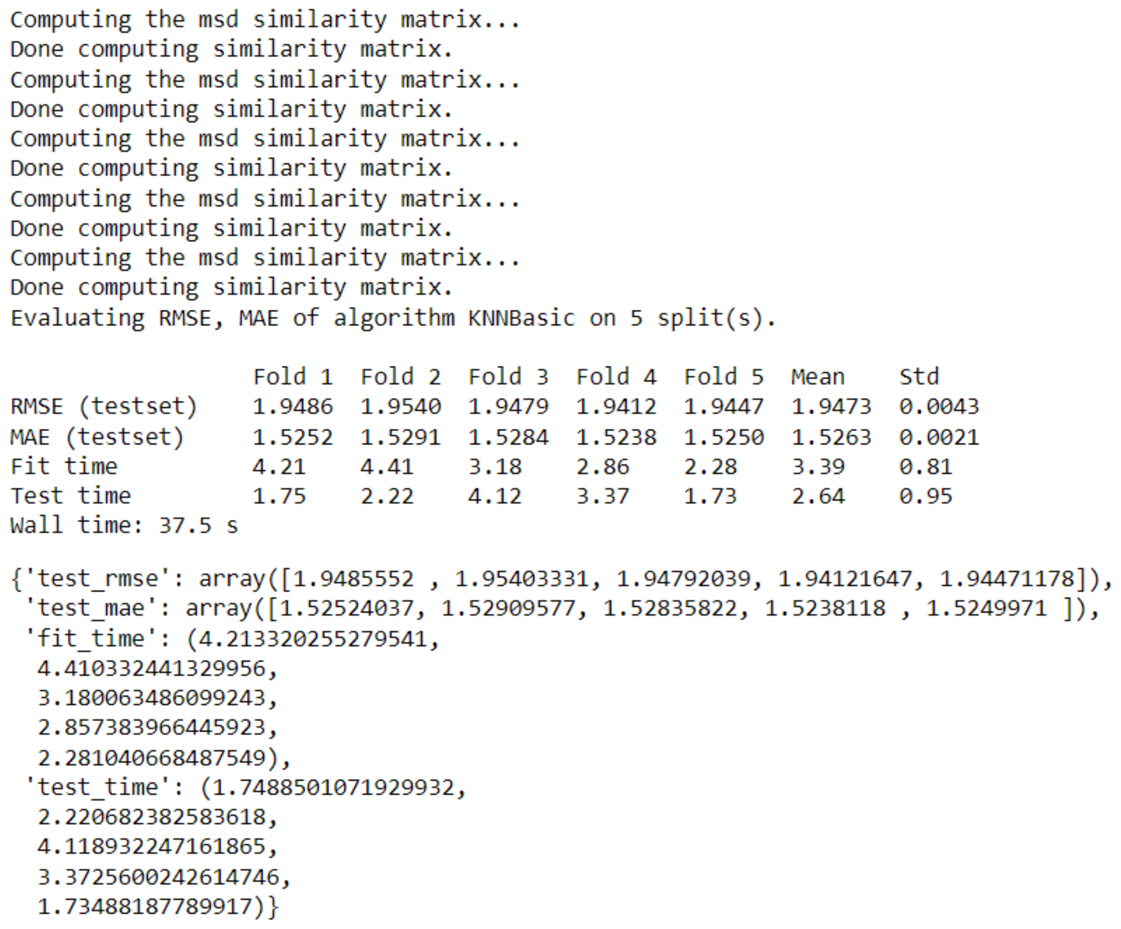


```
Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    2.4838  2.4745  2.4655  2.4794  2.4803  2.4767  0.0063
MAE (testset)     2.0771  2.0706  2.0626  2.0759  2.0725  2.0717  0.0051
Fit time          24.73   28.00   28.61   28.63   28.27   27.65   1.48
Test time         0.43    0.52    0.59    0.92    0.82    0.66    0.18
Wall time: 2min 26s

{'test_rmse': array([2.4837934 , 2.47453602, 2.46546762, 2.47943085, 2.48027558]),
 'test_mae': array([2.07709908, 2.07058525, 2.06256576, 2.07587563, 2.07245932]),
 'fit_time': (24.734967947006226,
  27.997292041778564,
  28.613962411880493,
  28.63470196723938,
  28.27423334121704),
 'test_time': (0.4348490238189697,
  0.5211153030395508,
  0.5895226001739502,
  0.9240365028381348,
  0.8198585510253906)}
```

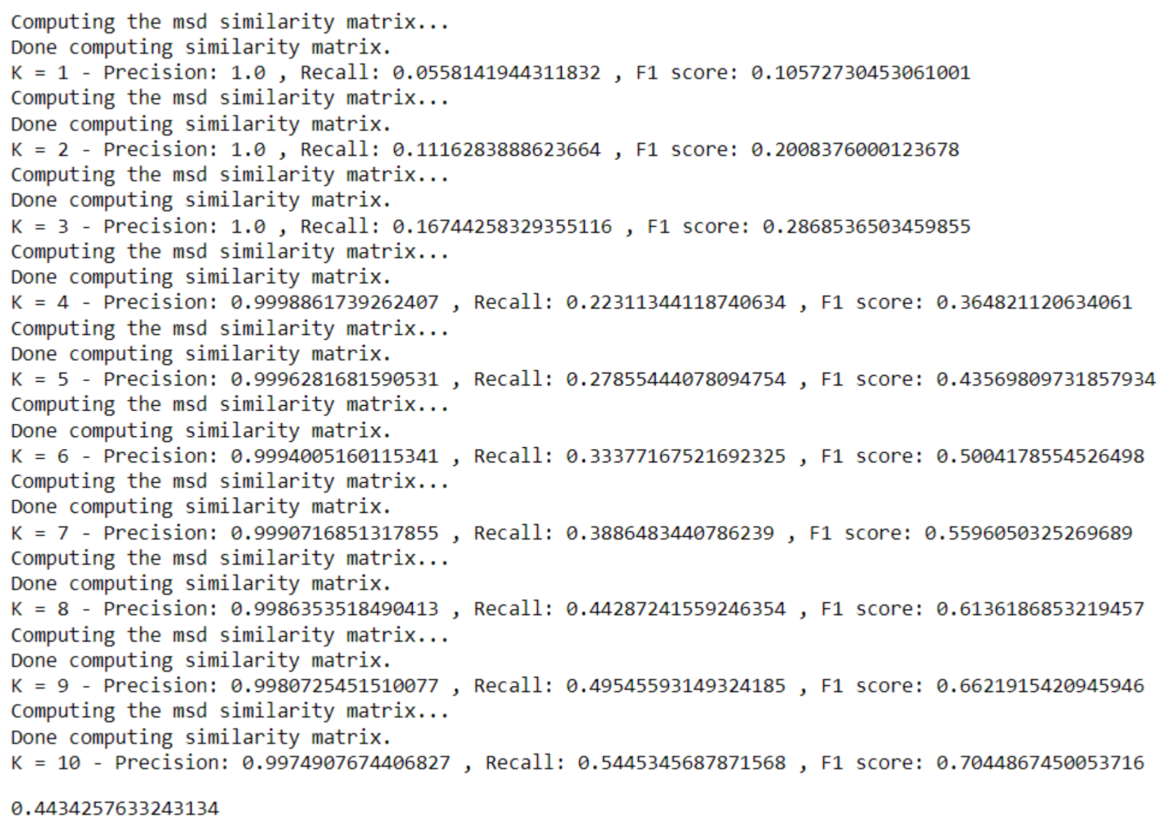
```
K = 1 - Precision: 1.0 , Recall: 0.05528344645974983 , F1 score: 0.1047745923528204
K = 2 - Precision: 1.0 , Recall: 0.10979920640237542 , F1 score: 0.19787220205051392
K = 3 - Precision: 1.0 , Recall: 0.16341350787902456 , F1 score: 0.2809207676760391
K = 4 - Precision: 1.0 , Recall: 0.21593751065467182 , F1 score: 0.3551786317347988
K = 5 - Precision: 1.0 , Recall: 0.2667822221019349 , F1 score: 0.421196662610675
K = 6 - Precision: 1.0 , Recall: 0.3156189593077581 , F1 score: 0.47980299626242534
K = 7 - Precision: 1.0 , Recall: 0.36162507534168226 , F1 score: 0.5311668856435199
K = 8 - Precision: 1.0 , Recall: 0.403863011662147 , F1 score: 0.5753595732734362
K = 9 - Precision: 1.0 , Recall: 0.4412786603154031 , F1 score: 0.612343292752056l
K = 10 - Precision: 1.0 , Recall: 0.47428741488401 , F1 score: 0.6434124175459026

0.4202028021902187
```

```
Evaluating RMSE, MAE of algorithm CoClustering on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.8294  1.8368  1.8379  1.8524  1.8427  1.8399  0.0076
MAE (testset)     1.4190  1.4263  1.4268  1.4363  1.4285  1.4274  0.0055
Fit time          18.42   18.41   18.32   19.94   18.63   18.75   0.61
Test time         0.81    0.76    0.34    0.38    0.36    0.53    0.21
Wall time: 1min 41s

{'test_rmse': array([1.82939069, 1.83683889, 1.83793975, 1.85242348, 1.84270441]),
 'test_mae': array([1.41899634, 1.4262525 , 1.42675985, 1.43634817, 1.42854398]),
 'fit_time': (18.415239572525024,
  18.413434982299805,
  18.32215166091919,
  19.942265033721924,
  18.633750200271606),
 'test_time': (0.8093860149383545,
  0.7604765892028809,
  0.33760499954223633,
  0.3770110607147217,
  0.35755085945129395)}
```

```
K = 1 - Precision: 0.9959022613446653 , Recall: 0.05478651848414059 , F1 score: 0.10385952281406677
K = 2 - Precision: 0.9945363484595539 , Recall: 0.10898860594753843 , F1 score: 0.19644889723585837
K = 3 - Precision: 0.9934233824050196 , Recall: 0.16255891342941192 , F1 score: 0.2793984409641183
K = 4 - Precision: 0.9932210249405576 , Recall: 0.21586193374910095 , F1 score: 0.3546466510722918
K = 5 - Precision: 0.9936990944503494 , Recall: 0.27012783154284636 , F1 score: 0.42478250157396646
K = 6 - Precision: 0.989958010826121 , Recall: 0.3190064752954811 , F1 score: 0.48252342836263723
K = 7 - Precision: 0.9918103043311739 , Recall: 0.3733489795382788 , F1 score: 0.5424881468307894
K = 8 - Precision: 0.989797750941324 , Recall: 0.4227116571832052 , F1 score: 0.5924194843164254
K = 9 - Precision: 0.9883283105319758 , Recall: 0.46745495804227194 , F1 score: 0.6347084472047584
K = 10 - Precision: 0.9878901704861583 , Recall: 0.5159641998835478 , F1 score: 0.6778794162926883

0.42891549366676013
```



```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.9486  1.9540  1.9479  1.9412  1.9447  1.9473  0.0043
MAE (testset)     1.5252  1.5291  1.5284  1.5238  1.5250  1.5263  0.0021
Fit time          4.21    4.41    3.18    2.86    2.28    3.39    0.81
Test time         1.75    2.22    4.12    3.37    1.73    2.64    0.95
Wall time: 37.5 s

{'test_rmse': array([1.9485552 , 1.95403331, 1.94792039, 1.94121647, 1.94471178]),
 'test_mae': array([1.52524037, 1.52909577, 1.52835822, 1.5238118 , 1.5249971 ]),
 'fit_time': (4.213320255279541,
  4.410332441329956,
  3.180063486099243,
  2.857383966445923,
  2.281040668487549),
 'test_time': (1.7488501071929932,
  2.220682382583618,
  4.118932247161865,
  3.3725600242614746,
  1.73488187789917)}
```



```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0558141944311832 , F1 score: 0.10572730453061001
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.1116283888623664 , F1 score: 0.2008376000123678
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16744258329355116 , F1 score: 0.2868536503459855
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 0.9998861739262407 , Recall: 0.22311344118740634 , F1 score: 0.364821120634061
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 0.9996281681590531 , Recall: 0.27855444078094754 , F1 score: 0.43569809731857934
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.9994005160115341 , Recall: 0.33377167521692325 , F1 score: 0.5004178554526498
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9990716851317855 , Recall: 0.3886483440786239 , F1 score: 0.5596050325269689
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9986353518490413 , Recall: 0.44287241559246354 , F1 score: 0.6136186853219457
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.9980725451510077 , Recall: 0.49545593149324185 , F1 score: 0.6621915420945946
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9974907674406827 , Recall: 0.544534568787l568 , F1 score: 0.7044867450053716

0.4434257633243134
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.8048  1.7915  1.8015  1.8066  1.7924  1.7994  0.0063
MAE (testset)     1.3961  1.3898  1.3944  1.3943  1.3879  1.3925  0.0031
Fit time          2.04    2.92    3.64    4.24    4.34    3.43    0.86
Test time         2.04    1.53    4.33    7.79    1.41    3.42    2.43
Wall time: 40.7 s

{'test_rmse': array([1.80477976, 1.79150214, 1.80151869, 1.80658759, 1.79244436]),
 'test_mae': array([1.39605786, 1.38975801, 1.39442556, 1.39426971, 1.38792112]),
 'fit_time': (2.0384974479675293,
  2.9201929569244385,
  3.637817144393921,
  4.236332654953003,
  4.335025072097778),
 'test_time': (2.0447285175323486,
  1.5310471057891846,
  4.325582265853882,
  7.792828798294067,
  1.4102652072906494)}
```

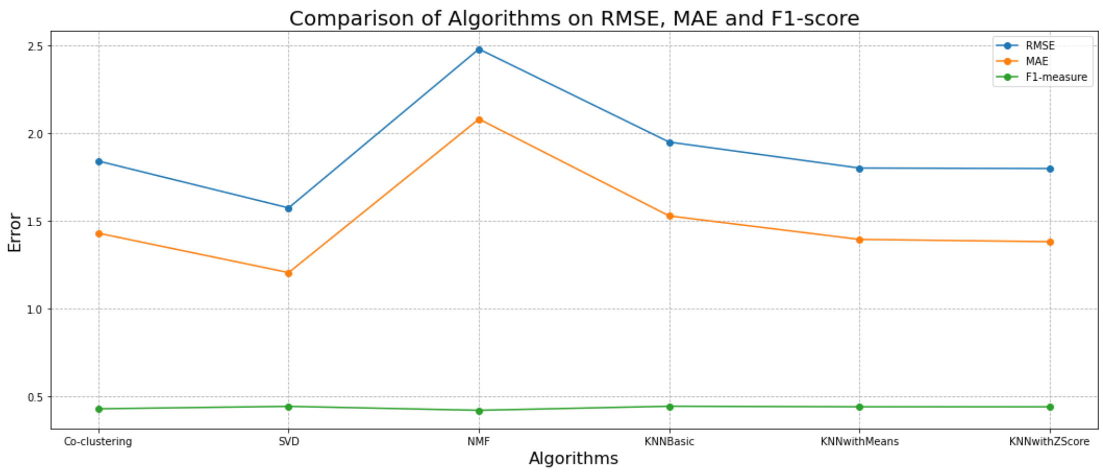
```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0557610755967622 , F1 score: 0.10563199740100972
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.11139784589392698 , F1 score: 0.20046439050694165
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16678232636527177 , F1 score: 0.28588421781263607
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 1.0 , Recall: 0.22189241975087412 , F1 score: 0.36319469073409055
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 0.9999696463803309 , Recall: 0.27676027864209213 , F1 score: 0.4335323744542044
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.999893762331158 , Recall: 0.3311963249486476 , F1 score: 0.49757885298343946
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9998142647558341 , Recall: 0.38508257213650193 , F1 score: 0.5560140488079885
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9997329604174345 , Recall: 0.43792602727380936 , F1 score: 0.6090583197248677
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.9994547189038002 , Recall: 0.48853384179334786 , F1 score: 0.6562785043129665
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9989488252426476 , Recall: 0.5354894162341955 , F1 score: 0.6972278307690934

0.4404865227507238
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithZScore on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.7955  1.7933  1.8084  1.7911  1.7950  1.7967  0.0061
MAE (testset)     1.3822  1.3807  1.3902  1.3761  1.3804  1.3819  0.0046
Fit time          3.66    3.17    2.24    2.01    3.22    2.86    0.63
Test time         1.56    1.63    1.39    1.40    1.55    1.51    0.10
Wall time: 30.2 s

{'test_rmse': array([1.79550758, 1.79327915, 1.80844816, 1.79107004, 1.7950232 ]),
 'test_mae': array([1.38223467, 1.3807193 , 1.39021685, 1.37613402, 1.3804225 ]),
 'fit_time': (3.6553030014038086,
  3.1679623126983643,
  2.2401254177093506,
  2.0087034702301025,
  3.222465753555298),
 'test_time': (1.562856674194336,
  1.6321866512298584,
  1.3893184661865234,
  1.400294303894043,
  1.5483851432800293)}
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0557610755967622 , F1 score: 0.10563199740100972
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.11137399662132981 , F1 score: 0.2004257737897703
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16677173216360153 , F1 score: 0.2858686537671745
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 1.0 , Recall: 0.221871165647058 , F1 score: 0.363166218968042
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 1.0 , Recall: 0.27674753717199757 , F1 score: 0.43351959430443826
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.9999494106338848 , Recall: 0.3310665599816539 , F1 score: 0.49743927772879526
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9998446183755032 , Recall: 0.3847135260378307 , F1 score: 0.5556339402245543
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9997795749047836 , Recall: 0.43720501067215817 , F1 score: 0.6083692812063879
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.99965520937975 , Recall: 0.4876773339649928 , F1 score: 0.6555483366190763
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9991999050847126 , Recall: 0.5345184458864329 , F1 score: 0.6964652669866893

0.4402068340995938
```

4. PERFORMANCE ANALYSIS

Algorithm	RMSE	F1 measure	MAE
Co-clustering	1.8393	0.4289	1.4274
SVD	1.5726	0.4428	1.2046
NMF	2.4767	0.4202	2.0717
KNNBasic	1.9473	0.4434	1.5263
KNNwithMeans	1.7994	0.4404	1.3925
KNNwithZScore	1.7967	0.4402	1.3819



5. CONCLUSION

Hence, this work concludes that the SVD technique is the most preferred among the algorithms implemented. Above figure shows that the NMF has a large RMSE and MAE and less F1 measurement compared to others. It further concludes that the NMF alone is not suitable for this dataset. Moreover, KNN (includes KNNBasic, KNNwithMeans, KNNwithZScore) is much better compared with NMF primarily based on RMSE and MAE values. In addition, it concludes that the evaluation of the RMSE is much better than that of the MAE.