**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**

**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**

* 1. 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.

* 1. 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.

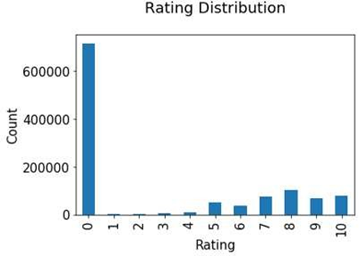


Figure 1: Rating Distribution of the books in dataset

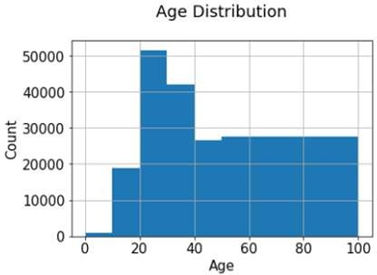


Figure 2: Age Distribution of users in user-data

**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.

* 1. 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.

1. **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 = 𝑋𝑆𝑌T

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).

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:

In the above RMSE equation, 𝑦j is the original data and 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:

In the above MAE equation, is the original data and 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. |

1. **IMPLEMENTATION** 
   1. 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.cumsum() / 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='coerce')

#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').reset\_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.cumsum() / 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\_list))

#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\_index(name = 'ratings\_counts')

df\_normalize = df\_normalize.sort\_values(by=['ratings\_counts'])

lazy\_users = df\_normalize.loc[(df\_normalize.ratings\_counts < 10),'user\_id'].unique().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\_rating']], 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 (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(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, threshold = 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\_rating']], reader)

model = SVD()

# Train on books dataset

%time model\_selection.cross\_validate(model, data, measures=['RMSE','MAE'],cv=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\_rating']], 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\_rating']], reader)

CoClusteringmodel = CoClustering()

%time model\_selection.cross\_validate(CoClusteringmodel, data, measures=['RMSE','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\_rating']], 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)

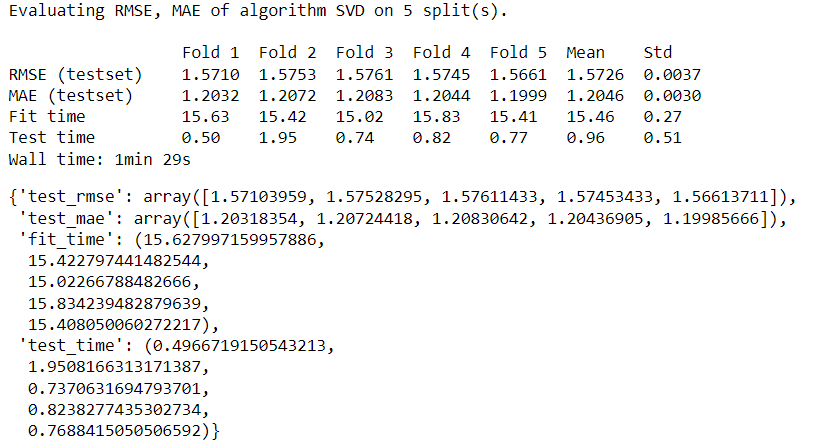
plt.ylabel('Error', fontsize=16)

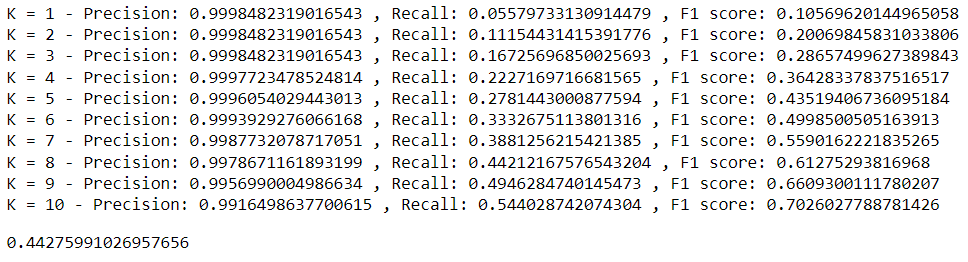
plt.grid(ls='dashed')

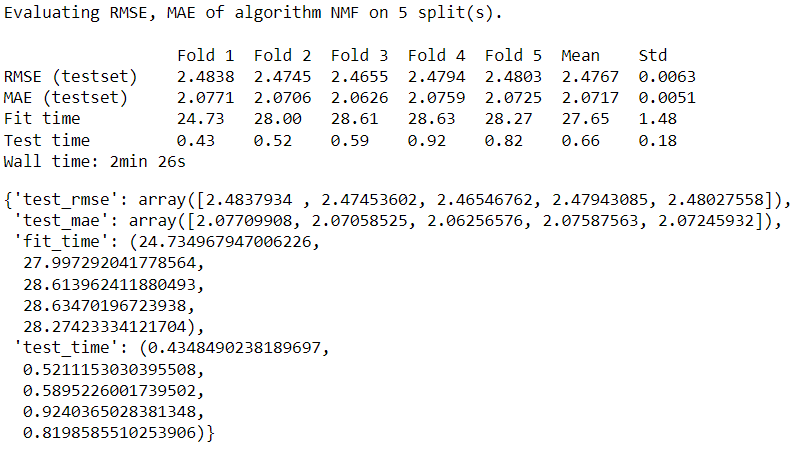
plt.legend()

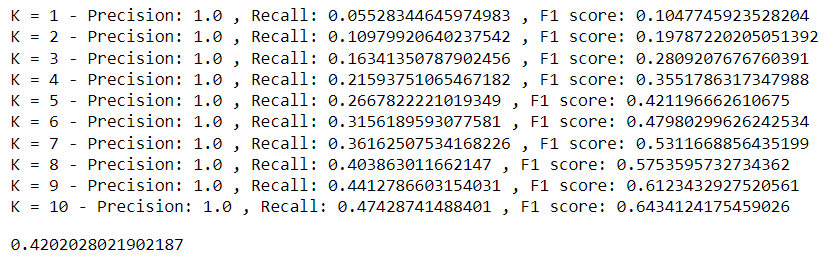
plt.show()

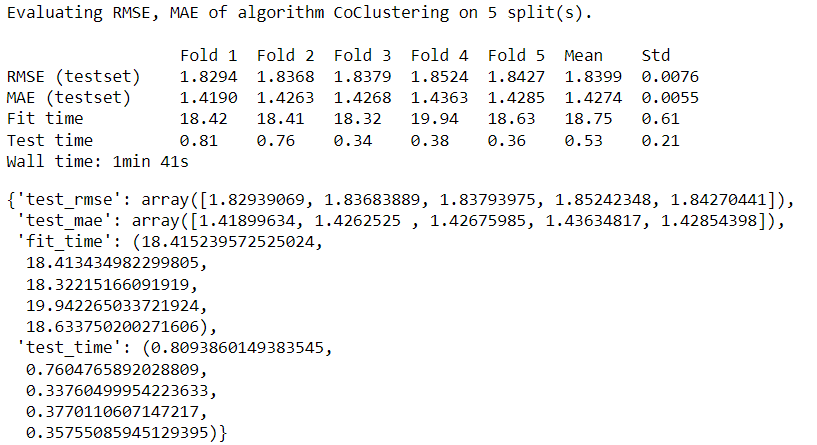
* 1. Screenshots

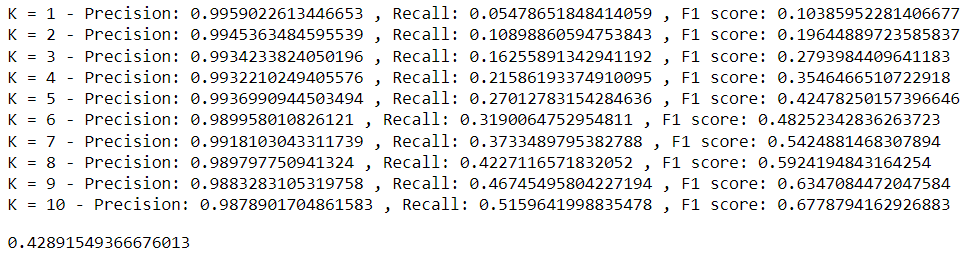


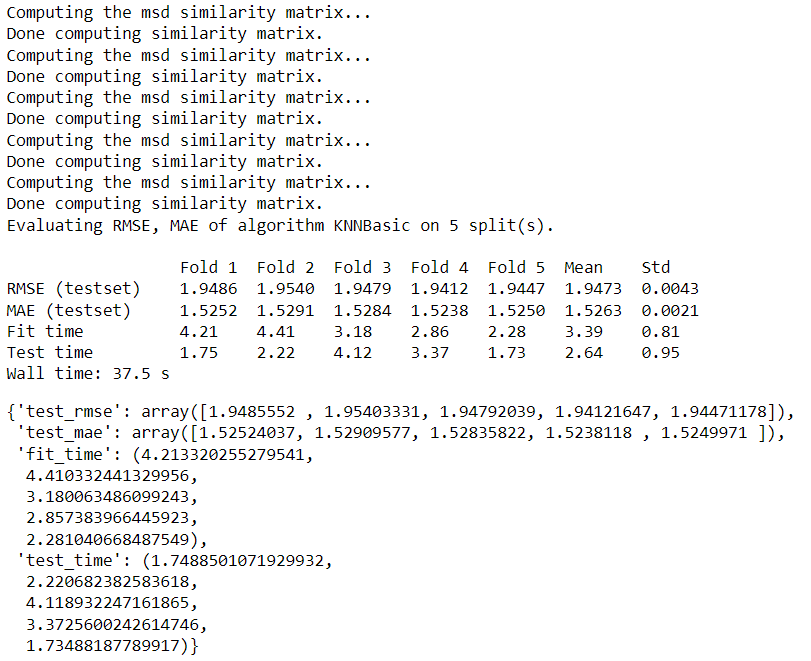


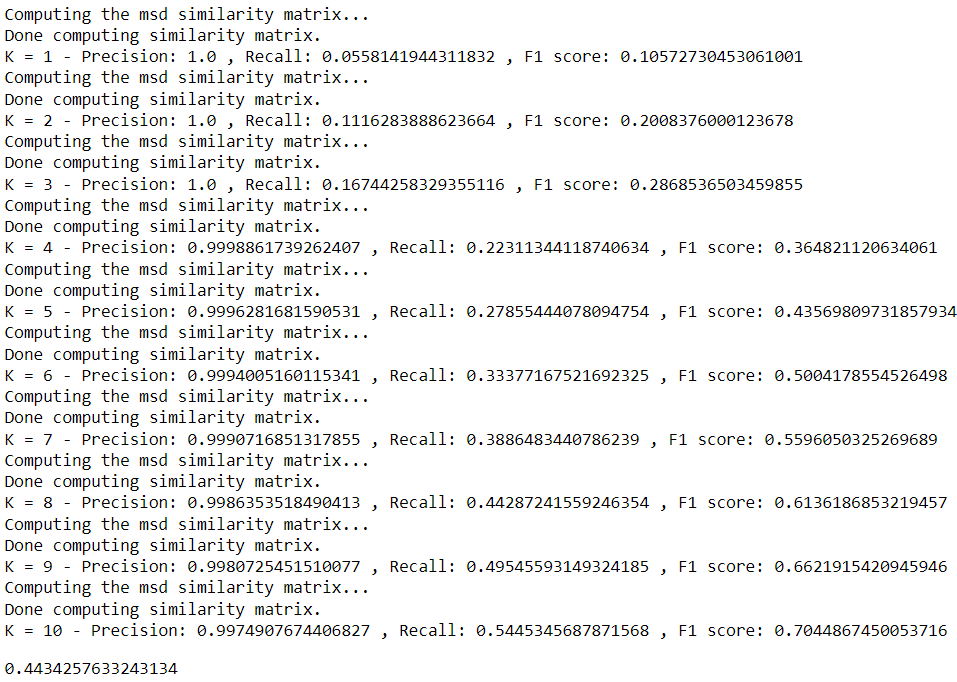


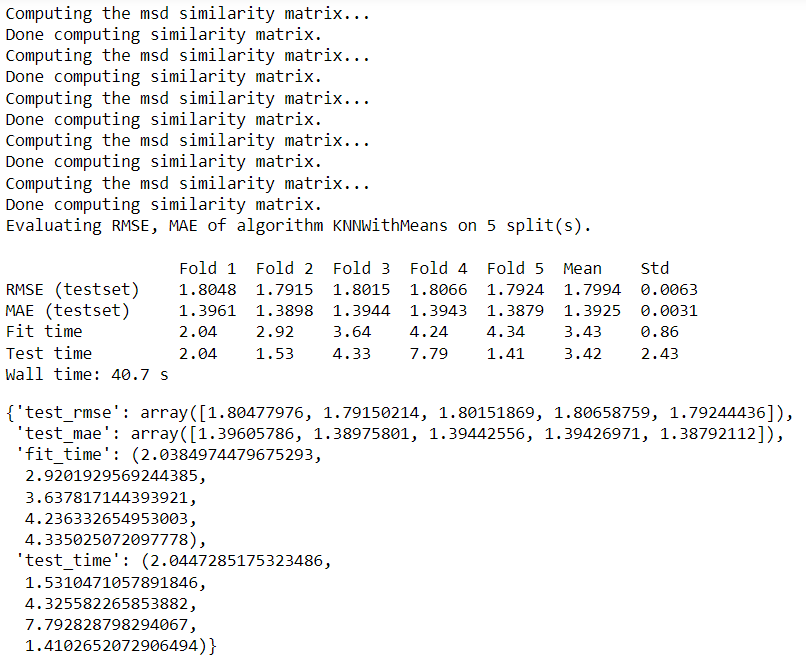


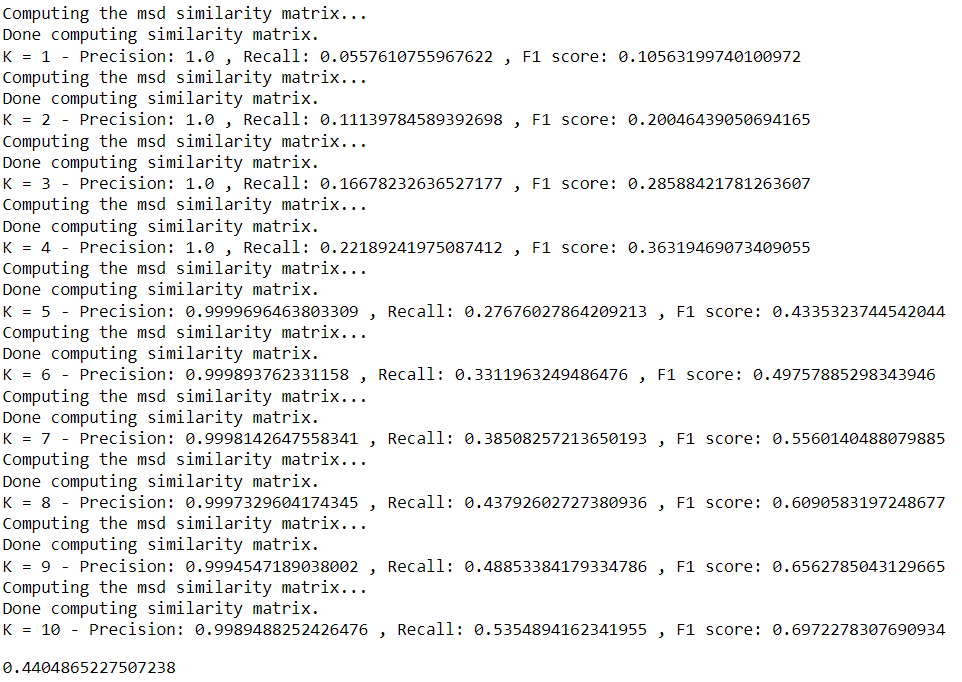
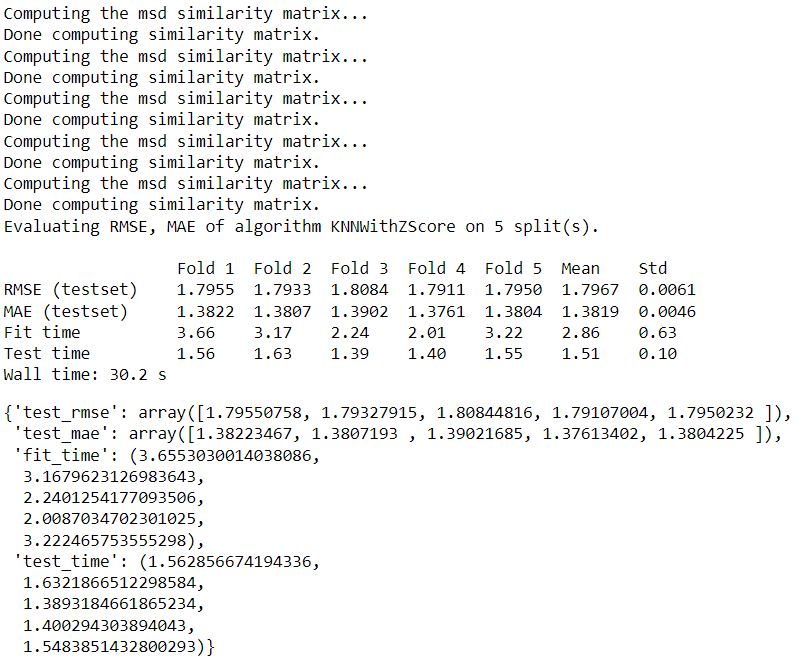


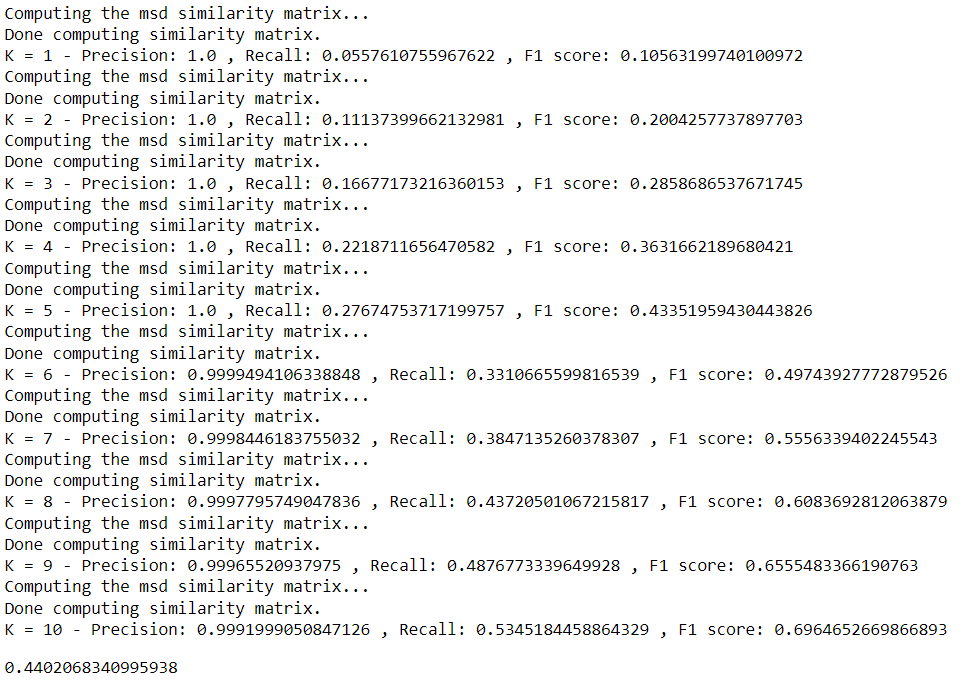






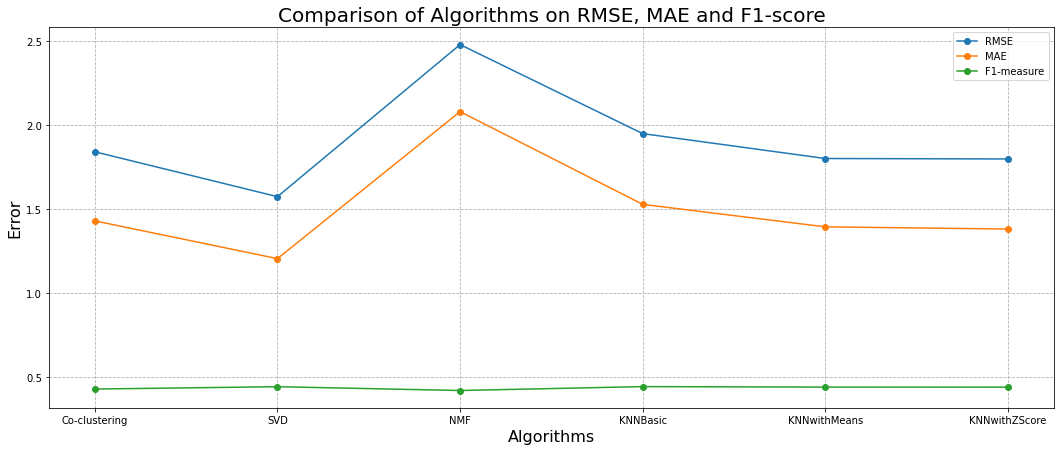




1. **PERFORMANCE ANALYSIS**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |



1. **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.