**Chapter 5**

**SYSTEM DESIGN**

**5.1 System Architecture**

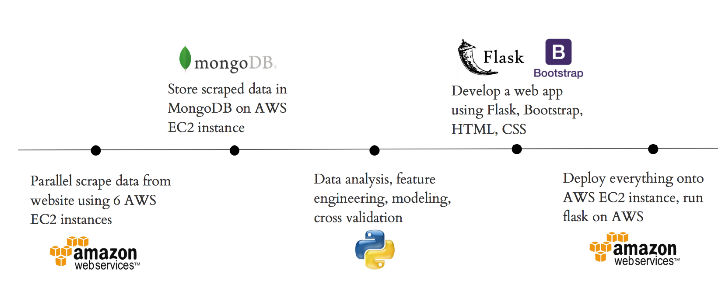


Figure 5.1 Final System Architecture

As shown in Figure 5.1, the data scraped using Scrapy will be stored in MongoDB from where it will be used by recommender algorithms and front-end technologies accordingly. The scraped data will first be cleaned and then an exploratory data analysis(EDA) will be performed on it to find the most famous restaurants, cuisines preferred, price range and other necessary trends in the data.

Once the EDA is done on the scraped data, the data will further be analysed. Feature engineering will then be performed on it followed by modelling and cross validation. A web application will be developed using Flask, Bootstrap, HTML and CSS.

**Chapter 6**

**IMPLEMENTATION**

**6.1 Pseudo-code / Algorithm**

To achieve minimal RMSE, **Singular Value Decomposition** (SVD) algorithm is adopted. One way to handle the scalability and sparsity issue created by collaborative filtering is to leverage a **latent factor model** to capture the similarity between users and items. Essentially, it is required to turn the recommendation problem into an optimization problem. It can be viewed as how good the machine is in predicting the rating for items given a user. One common metric is **Root Mean Square Error**(RMSE). The lower the RMSE, the better the performance. Since the rating for the unseen items is not known, it will temporarily ignore them. Namely, RMSE will be minimized on the known entries in the utility matrix.  SVD decreases the dimension of the utility matrix by extracting its latent factors. SVD has a great property that it has the minimal reconstruction Sum of Square Error (SSE); therefore, it is also commonly used in dimensionality reduction. However, SVD is not without flaw. The main drawback of SVD is that there is no to little explanation to the reason that we recommend an item to a user. This can be a huge problem if users are eager to know why a specific item is recommended to them. The following code snippet shows the implementation of the SVD algorithm.

def svd(A, k=None, epsilon=1e-10):

A = np.array(A, dtype=float)

n, m = A.shape

svdSoFar = []

if k is None:

k = min(n, m)

for i in range(k):

matrixFor1D = A.copy()

for singularValue, u, v in svdSoFar[:i]:

matrixFor1D -= singularValue \* np.outer(u, v)

if n > m:

v = svd\_1d(matrixFor1D, epsilon=epsilon) # next singular vector

u\_unnormalized = np.dot(A, v)

sigma = norm(u\_unnormalized) # next singular value

u = u\_unnormalized / sigma

else:

u = svd\_1d(matrixFor1D, epsilon=epsilon) # next singular vector

v\_unnormalized = np.dot(A.T, u)

sigma = norm(v\_unnormalized) # next singular value

v = v\_unnormalized / sigma

svdSoFar.append((sigma, u, v))

singularValues, us, vs = [np.array(x) for x in zip(\*svdSoFar)]

return singularValues, us.T, vs

The goal of a recommender system is to improve the predictive accuracy. In fact, the user will leave some implicit feedback information, such as historical browsing data, and historical rating data, on Web applications as long as any user has rated item no matter what the specific rating value is. To a certain extent, the rating operation already reflects the degree of a user’s preference for each latent factor. Therefore, the **SVD++ model** introduces the implicit feedback information based on SVD; that is, it adds a factor vector () for each item, and these item factors are used to describe the characteristics of the item, regardless of whether it has been evaluated. Then, the user’s factor matrix is modelled, so that a better user bias can be obtained. The SVD++ process can be divided into the following four stages:

1. Inputting of the original rating matrix
2. SVD++ factorization process by SGD or ALS
3. Outputting of the user characteristic matrix and the item characteristic matrix
4. Rating prediction (i.e., recommendation)

The following code snippet shows the implementation of the SVD++ algorithm.

def svdpp(algo, trainset, testset, verbose=True):

# dictionaries that stores metrics for train and test..

train = dict()

test = dict()

# train the algorithm with the trainset

print('Training the model...')

algo.fit(trainset)

# ---------------- Evaluating train data--------------------#

print('Evaluating the model with train data..')

# get the train predictions (list of prediction class inside Surprise)

train\_preds = algo.test(trainset.build\_testset())

# get predicted ratings from the train predictions..

train\_actual\_ratings, train\_pred\_ratings = get\_ratings(train\_preds)

# get ''rmse'' and ''mape'' from the train predictions.

train\_rmse, train\_mape = get\_errors(train\_preds)

if verbose:

print('-'\*15)

print('Train Data')

print('-'\*15)

print("RMSE : {}\n\nMAPE : {}\n".format(train\_rmse, train\_mape))

#store them in the train dictionary

if verbose:

print('adding train results in the dictionary..')

train['rmse'] = train\_rmse

train['mape'] = train\_mape

train['predictions'] = train\_pred\_ratings

#------------ Evaluating Test data---------------#

print('\nEvaluating for test data...')

# get the predictions( list of prediction classes) of test data

test\_preds = algo.test(testset)

# get the predicted ratings from the list of predictions

test\_actual\_ratings, test\_pred\_ratings = get\_ratings(test\_preds)

# get error metrics from the predicted and actual ratings

test\_rmse, test\_mape = get\_errors(test\_preds)

**6.2 Configuration**

**Anaconda :**

Anaconda is a [free and open source](https://en.wikipedia.org/wiki/Free_and_open_source) distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [data science](https://en.wikipedia.org/wiki/Data_science) and learning related applications (large-scale data processing, [predictive](https://en.wikipedia.org/wiki/Predictive_analytics) [analytics,](https://en.wikipedia.org/wiki/Predictive_analytics) [scientific computing),](https://en.wikipedia.org/wiki/Scientific_computing) that aims to simplify management and deployment. Anaconda3 includes Python 3.6. Anaconda Navigator is a desktop [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface) [(GUI)](https://en.wikipedia.org/wiki/Graphical_user_interface) included in Anaconda distribution that allows users to launch applications and manage anaconda packages, environments and channels without using [command-line commands.](https://en.wikipedia.org/wiki/Command-line_interface) Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for [Windows,](https://en.wikipedia.org/wiki/Windows) [macOS](https://en.wikipedia.org/wiki/MacOS) and [Linux.](https://en.wikipedia.org/wiki/Linux)

The following are the system requirements:

* **License:** Free use and redistribution under the terms of the [Anaconda End User](https://docs.anaconda.com/anaconda/eula/) [License Agreement.](https://docs.anaconda.com/anaconda/eula/)
* **Operating system:** Windows Vista or newer, 64-bit macOS 10.10+, or Linux,including Ubuntu, RedHat, CentOS 6+, and others. Windows XP supported on Anaconda versions 2.2 and earlier. See lists. Download it from our [archive.](https://repo.anaconda.com/archive/)
* **System architecture:** 64-bit x86, 32-bit x86 with Windows or Linux, Power8 orPower9. Minimum 3 GB disk space to download and install.

**6.3 Code**

### Creating sparse matrix from test data frame :

if os.path.isfile('test\_sparse\_matrix.npz'):

print("It is present in your pwd, getting it from disk....")

# just get it from the disk instead of computing it

test\_sparse\_matrix = sparse.load\_npz('test\_sparse\_matrix.npz')

print("DONE..")

else:

print("We are creating sparse\_matrix from the dataframe..")

# create sparse\_matrix and store it for after usage.

# csr\_matrix(data\_values, (row\_index, col\_index), shape\_of\_matrix)

# It should be in such a way that, MATRIX[row, col] = data

test\_sparse\_matrix = sparse.csr\_matrix((test\_df.rating.values, (test\_df.userID.values,

test\_df.placeID.values)))

print('Done. It\'s shape is : (user, place) : ',test\_sparse\_matrix.shape)

print('Saving it into disk for further usage..')

# save it into disk

sparse.save\_npz("test\_sparse\_matrix.npz", test\_sparse\_matrix)

print('Done..\n')

### PDF's and CDF's of avg ratings of users and places (train data) :

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=plt.figaspect(.5))

fig.suptitle('Avg Ratings per User and per Place', fontsize=15)

ax1.set\_title('Users-Avg-Ratings')

# get the list of average user ratings from the averages dictionary..

user\_averages = [rat for rat in train\_averages['userID'].values()]

sns.distplot(user\_averages, ax=ax1, hist=False,

kde\_kws=dict(cumulative=True), label='Cdf')

sns.distplot(user\_averages, ax=ax1, hist=False,label='Pdf')

ax2.set\_title('Places-Avg-Rating')

# get the list of place\_average\_ratings from the dictionary..

place\_averages = [rat for rat in train\_averages['placeID'].values()]

sns.distplot(place\_averages, ax=ax2, hist=False,

kde\_kws=dict(cumulative=True), label='Cdf')

sns.distplot(place\_averages, ax=ax2, hist=False, label='Pdf')

plt.show()

**Cold Start Problem:**

total\_users = len(np.unique(rating\_final\_df.userID))

users\_train = len(train\_averages['userID'])

new\_users = total\_users - users\_train

print('\nTotal number of Users :', total\_users)

print('\nNumber of Users in Train data :', users\_train)

print("\nNo of Users that didn't appear in train data: {}({} %) \n ".format(new\_users,

np.round((new\_users/total\_users)\*100, 2)))

total\_places = len(np.unique(rating\_final\_df.placeID))

places\_train = len(train\_averages['placeID'])

new\_places = total\_places - places\_train

print('\nTotal number of Places :', total\_places)

print('\nNumber of Users in Train data :', places\_train)

print("\nNo of Places that didn't appear in train data: {}({} %) \n ".format(new\_places,

np.round((new\_places/total\_places)\*100, 2)))

**Utility modules required for algorithm implementation:**

from surprise import Reader, Dataset

#Transforming train data

# It is to specify how to read the dataframe.

# for our dataframe, we don't have to specify anything extra..

reader = Reader(rating\_scale=(1,5))

# create the traindata from the dataframe...

train\_data = Dataset.load\_from\_df(reg\_train[['userID', 'placeID', 'rating']], reader)

# build the trainset from traindata.., It is of dataset format from surprise library..

trainset = train\_data.build\_full\_trainset()

#Transforming test data

testset = list(zip(reg\_test\_df.userID.values, reg\_test\_df.placeID.values, reg\_test\_df.rating.values))

testset[:3]

def get\_ratings(predictions):

actual = np.array([pred.r\_ui for pred in predictions])

pred = np.array([pred.est for pred in predictions])

return actual, pred

def get\_errors(predictions, print\_them=False):

actual, pred = get\_ratings(predictions)

rmse = np.sqrt(np.mean((pred - actual)\*\*2))

mape = np.mean(np.abs(pred - actual)/actual)

return rmse, mape\*100

def run\_surprise(algo, trainset, testset, verbose=True):

'''

return train\_dict, test\_dict

It returns two dictionaries, one for train and the other is for test

Each of them have 3 key-value pairs, which specify ''rmse'', ''mape'', and ''predicted ratings''.

'''

# dictionaries that stores metrics for train and test..

train = dict()

test = dict()

# train the algorithm with the trainset

print('Training the model...')

algo.fit(trainset)

# ---------------- Evaluating train data--------------------#

print('Evaluating the model with train data..')

# get the train predictions (list of prediction class inside Surprise)

train\_preds = algo.test(trainset.build\_testset())

# get predicted ratings from the train predictions..

train\_actual\_ratings, train\_pred\_ratings = get\_ratings(train\_preds)

# get ''rmse'' and ''mape'' from the train predictions.

train\_rmse, train\_mape = get\_errors(train\_preds)

if verbose:

print('-'\*15)

print('Train Data')

print('-'\*15)

print("RMSE : {}\n\nMAPE : {}\n".format(train\_rmse, train\_mape))

if verbose:

print('adding train results in the dictionary..')

train['rmse'] = train\_rmse

train['mape'] = train\_mape

train['predictions'] = train\_pred\_ratings

#------------ Evaluating Test data---------------#

print('\nEvaluating for test data...')

# get the predictions( list of prediction classes) of test data

test\_preds = algo.test(testset)

# get the predicted ratings from the list of predictions

test\_actual\_ratings, test\_pred\_ratings = get\_ratings(test\_preds)

# get error metrics from the predicted and actual ratings

test\_rmse, test\_mape = get\_errors(test\_preds)

if verbose:

print('-'\*15)

print('Test Data')

print('-'\*15)

print("RMSE : {}\n\nMAPE : {}\n".format(test\_rmse, test\_mape))

# store them in test dictionary

if verbose:

print('storing the test results in test dictionary...')

test['rmse'] = test\_rmse

test['mape'] = test\_mape

test['predictions'] = test\_pred\_ratings

print('\n'+'-'\*45)

# return two dictionaries train and test

return train, test

**SVD Matrix Factorization with User Place interactions :**

from surprise import SVD

# Initialize the model

svd = SVD(n\_factors=100, biased=True, random\_state=15, verbose=True)

svd\_train\_results, svd\_test\_results = run\_surprise(svd, trainset, testset, verbose=True)

# Just store these error metrics in our models\_evaluation datastructure

models\_evaluation\_train['svd'] = svd\_train\_results

models\_evaluation\_test['svd'] = svd\_test\_results

### SVD Matrix Factorization with implicit feedback from user

from surprise import SVDpp

# initiallize the model

svdpp = SVDpp(n\_factors=50, random\_state=15, verbose=True)

svdpp\_train\_results, svdpp\_test\_results = run\_surprise(svdpp, trainset, testset, verbose=True)

# Just store these error metrics in our models\_evaluation datastructure

models\_evaluation\_train['svdpp'] = svdpp\_train\_results

models\_evaluation\_test['svdpp'] = svdpp\_test\_results

# Comparison and efficiency between the two models :

pd.DataFrame(models\_evaluation\_test).to\_csv('small\_sample\_results.csv')

models = pd.read\_csv('small\_sample\_results.csv', index\_col=0)

models.loc['rmse'].sort\_values()

**Chapter 7**

**RESULT ANALYSIS**

**7.1 Testing**

Software Testing is the process used to help identify the correctness, completeness, security and quality of the developed computer software. Testing is the process of technical investigation and includes the process of executing a program or application with the intent of finding errors.

**Unit Testing :**

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. Unit testing has been done to check whether the cancer dataset is loaded correctly.

The table 7.1 checks whether the dataset is loaded accordingly as the output shown in figure 7.1.

Table 7.1 Unit test case for Dataset Loading

|  |  |
| --- | --- |
| Sl No. of test case: | 1 |
|  |  |
| Name of test: | Check test |
|  |  |
| Item / Feature being tested: | Input Dataset |
|  |  |
| Sample Input: | Syntax to display top 5 element from dataset specified in |
|  | a particular directory |
|  |  |
| Expected output: | Displays top 5 elements |
|  |  |
| Expected output: | Displays top 5 elements |
|  |  |
| Remarks | Test Succeeded |
|  |  |

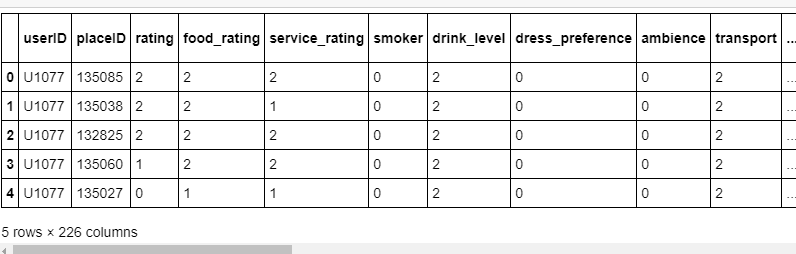


Figure 7.1 Dataset loaded

In table 7.2, Checking is done for null/NA value after dropping.

Table 7.2 Unit test to check for null/NA value

|  |  |
| --- | --- |
| Sl No. of test case: | 2 |
|  |  |
| Name of test: | Check test |
|  |  |
| Item / Feature being tested: | Dataframe object |
|  |  |
| Sample Input: | Drop command to drop null/NA values |
|  |  |
| Expected output: | No null/NA value in dataframe |
|  |  |
| Expected output: | No null/NA value in dataframe |
|  |  |
| Remarks | Test Succeeded |
|  |  |

**Integration Testing :**

Integration testing is done to test the modules/components when integrated to verify that they work as expected i.e. to test the modules which are working fine individually does not have issues when integrated. It is used to check the train and test split function as shown in table 7.3 and the output seen in figure 7.2 as shown below.

Table 7.3 Integration test to check for train-test-split

|  |  |
| --- | --- |
| Sl No. of test case: | 1 |
|  |  |
| Name of test: | Check test |
|  |  |
| Item / Feature being tested: | Train-test-split function |
|  |  |
| Sample Input: | Input features and ratio and check for the correct splitting |
|  | of the dataframe |
|  |  |
| Expected output: | Split according to the given ratio |
|  |  |
| Expected output: | Splitted according to the given ratio |
|  |  |
| Remarks | Test Succeeded |
|  |  |
|  |  |

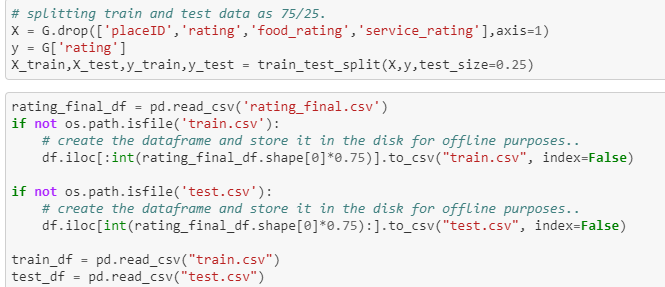


Figure 7.2 Splitting the dataset into the ratio 3:1

**System Testing :**

System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. The software or hardware is testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements. System testing takes, as its input, all of the ‘integrated’ software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s).

In table 7.4, one is checking accuracy using logistic regression algorithm and the output is shown in figure 7.3 as shown below.

Table 7.4 System test to compare the accuracy of SVD and SVD++

|  |  |
| --- | --- |
| Sl No. of test case: | 1 |
|  |  |
| Name of test: | Check test |
|  |  |
| Item / Feature being tested: | Comparing algorithm’s efficiency |
|  |  |
| Sample Input: | Training features and output feature |
|  |  |
| Expected output: | Some accuracy of the training model |
|  |  |
| Expected output: | SVD accuracy: 96.04% |
|  | SVD++ accuracy: 92.10% |
|  |  |
| Remarks | Test Succeeded |
|  |  |

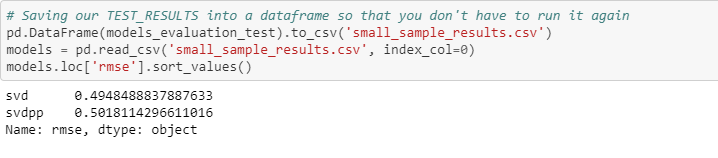


Figure 7.3 Comparing the accuracy between SVD and SVD++

**7.2 Results and Snapshots**

The following Figure 7.4 shows a graph depicting the most accepted payment mode. It is clear that most of the customers prefer to pay by cash. The second famous is MasterCard-Eurocard and VISA.

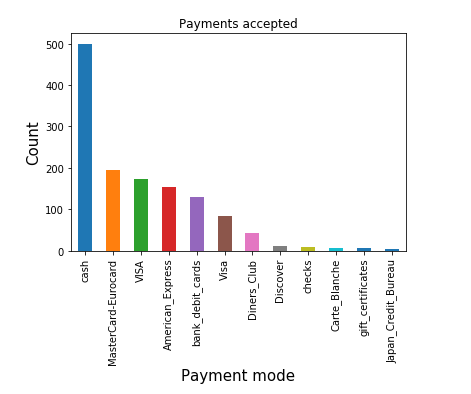


Figure 7.4 Graph depicting the most accepted payment mode

Figure 7.5 represents the top cuisines offered by the restaurants. It is clear that Mexican beats the other cuisines by a large margin.

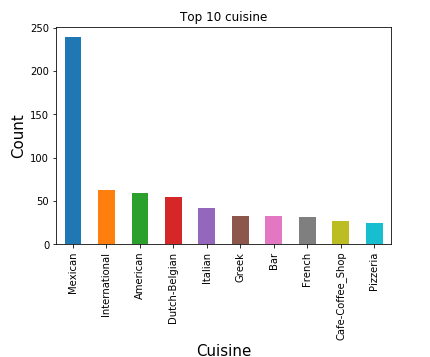


Figure 7.5 Graph depicting the top cuisines

Figure 7.6 represents the top favourite cuisines for the customers. It is clear that Mexican beats the other cuisines by a large margin. American is the second favourite cuisine for customers.

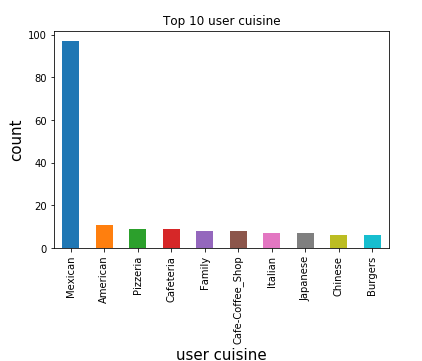


Figure 7.6 Graph depicting the top favourite cuisines for customers

Figure 7.7 shows the graph of user’s personal information based on birth year. It is clear that people born in 1989-92 have high interests and religion views.

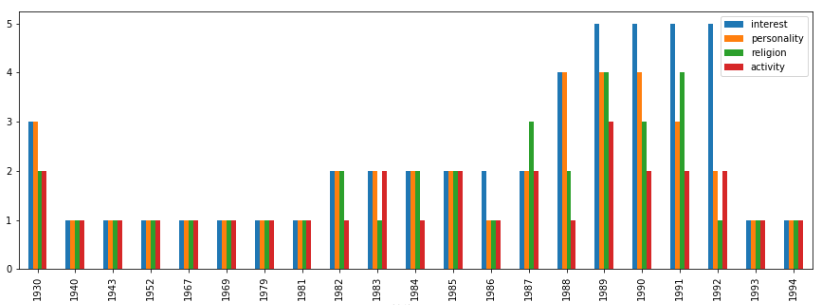


Figure 7.7 Graph depicting user’s personal information based on birth year

**Chapter 8**

**CONCLUSION AND FUTURE WORK**

**8.1 Conclusion**

The project is successful in identifying the better of the available two algorithms for recommendation – SVD, SVD++. It performs all the necessary data cleaning steps by checking the dataset for outliers, null values.

This project is a good introduction to training and test sets which are very important components not just to data science but to statistical learning overall. A complete exploratory data analysis(EDA) is performed on the dataset and various trends are found in the data, for instance, the most favourite cuisine among customers etc.,

**8.2 Limitations and future work**

**Limitations :**

Though the project focuses on SVD and SVD++, various other recommender system models can be used for comparison to find the algorithm with best accuracy and least RMSE.

**Future work :**

The future work for Phase II would involve scraping data from Zomato website instead of using a ready dataset. The dataset used for this project is a Mexican restaurant’s dataset. The scraped data is to be stored in MongoDB. The next task would be to build a web application for user interface instead of just seeing the results in the command line.

**REFERENCES**

[1] Lü L, Medo M, Yeung C H, et al. Recommender systems[J]. Physics Reports, 2012, 519(1): 1-49.USA: Abbrev. of Publisher, year, ch. x,sec. x, pp. xxx–xxx

[2] Recommender Systems, Aggarwal

[3] Achin Jain, Vanita Jain, Nidhi Kapoor ″A literature survey on recommendation system based on sentimental analysis″ Advanced Computational Intelligence, Vol.3, No.1, pp. 25-36, 2016.

[4] Assad Abbas, Limin Zhang, Samee U. Khan ″A survey on context-aware recommender systems based on computational intelligence techniques″, Computing, Vol.97, pp. 667–890, 2015

[5] Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan, Collaborative Filtering Recommender Systems, Foundations and Trends in Human Computer Interaction, vol. 4, 2011, pp 81-173

[6] Sachin Walunj, Kishor Sadafale. An online Recommendation System for E-commerce Based On Apache Mahout Framework. SIGMIS-CPR. ACM 2013, pp 153-158.