Chapter 5

SYSTEM DESIGN

5.1 System Architecture

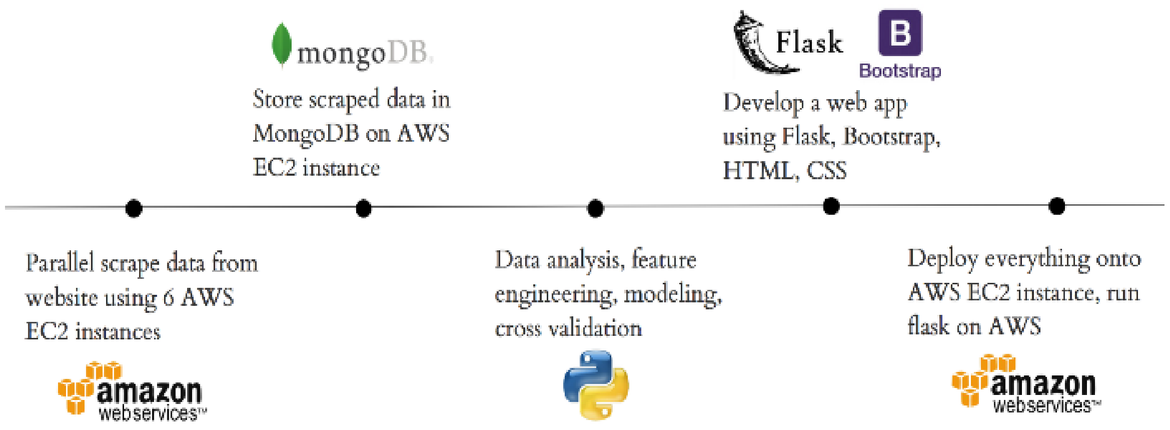

mongoDB
Store scraped data in MongoDB on AWS EC2 instance
Flask
Bootstrap
Develop a web app using Flask, Bootstrap, HTML, CSS
Parallel scrape data from website using 6 AWS EC2 instances
amazon webservices
Data analysis, feature engineering, modeling, cross validation
Deploy everything onto AWS EC2 instance, run flask on AWS
amazon webservices

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:

- (i) Inputting of the original rating matrix
- (ii) SVD++ factorization process by SGD or ALS
- (iii) Outputting of the user characteristic matrix and the item characteristic matrix
- (iv) 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 distribution of the Python and R programming languages for data science and learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify management and deployment. Anaconda3 includes Python 3.6. Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage anaconda packages, environments and channels without using command-line commands. 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, macOS and Linux.

The following are the system requirements:

- **License:** Free use and redistribution under the terms of the Anaconda End User License Agreement.

- **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.
- **System architecture:** 64-bit x86, 32-bit x86 with Windows or Linux, Power8 or Power9. 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.val
ues))
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 ratin
gs".
    '''
    # 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.

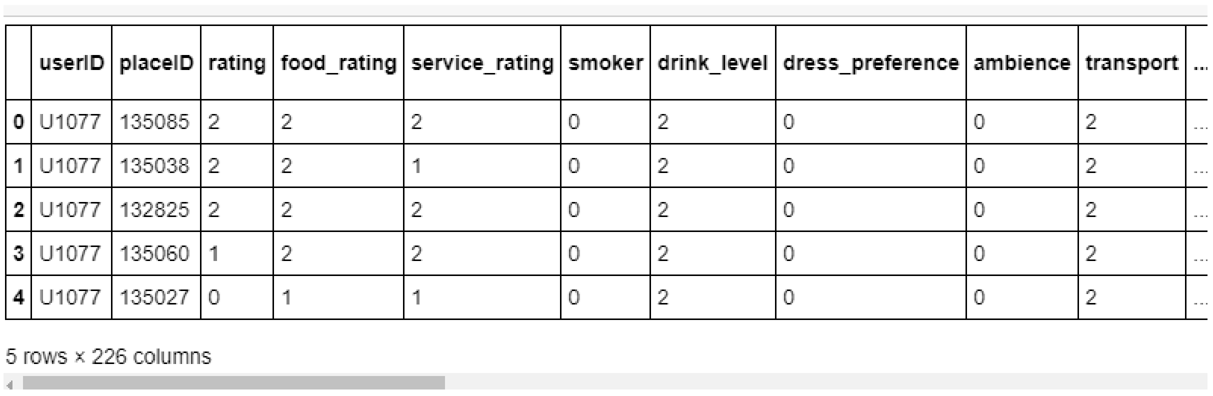
Unit Testing :

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



	userID	placeID	rating	food_rating	service_rating	smoker	drink_level	dress_preference	ambience	transport	...
0	U1077	135085	2	2	2	0	2	0	0	2	...
1	U1077	135038	2	2	1	0	2	0	0	2	...
2	U1077	132825	2	2	2	0	2	0	0	2	...
3	U1077	135060	1	2	2	0	2	0	0	2	...
4	U1077	135027	0	1	1	0	2	0	0	2	...

5 rows × 226 columns

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

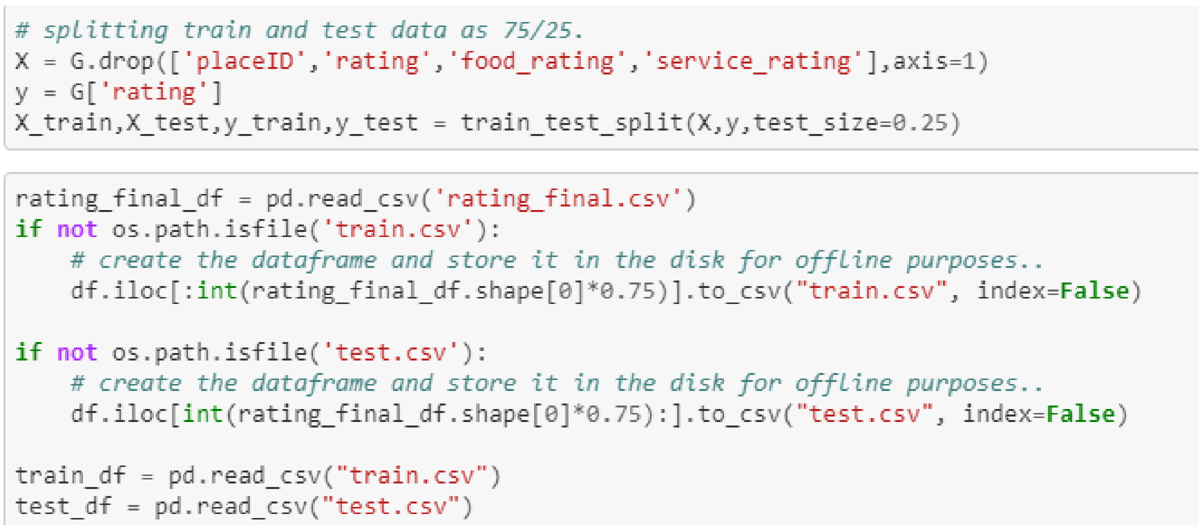
Sl No. of test case:	2
Name of test:	Check test
Item / Feature being tested:	Dataframe object
Sample Input:	Drop command to drop null/NA values
Expected output:	No null/NA value in dataframe
Expected output:	No null/NA value in dataframe
Remarks	Test Succeeded

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



```
# splitting train and test data as 75/25.
X = G.drop(['placeID','rating','food_rating','service_rating'],axis=1)
y = G['rating']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)
```

```
rating_final_df = pd.read_csv('rating_final.csv')
if not os.path.isfile('train.csv'):
    # create the dataframe and store it in the disk for offline purposes..
    df.iloc[:int(rating_final_df.shape[0]*0.75)].to_csv("train.csv", index=False)

if not os.path.isfile('test.csv'):
    # create the dataframe and store it in the disk for offline purposes..
    df.iloc[int(rating_final_df.shape[0]*0.75):].to_csv("test.csv", index=False)

train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

Figure 7.2 Splitting the dataset into the ratio 3:1

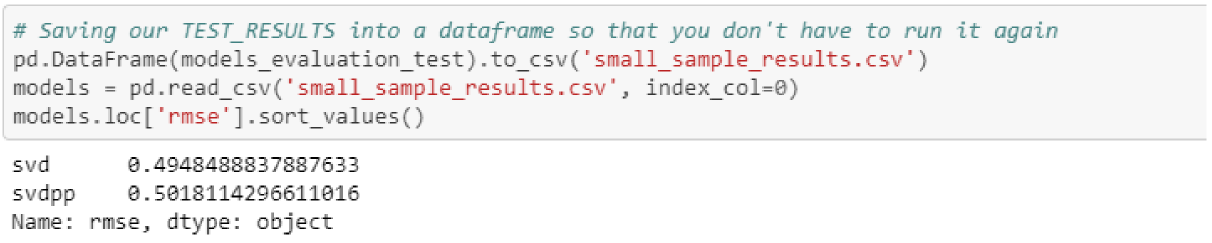
System Testing :

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



```
# Saving our TEST_RESULTS into a dataframe so that you don't have to run it again
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()
```

```
svd      0.4948488837887633
svdpp    0.5018114296611016
Name: rmse, dtype: object
```

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.

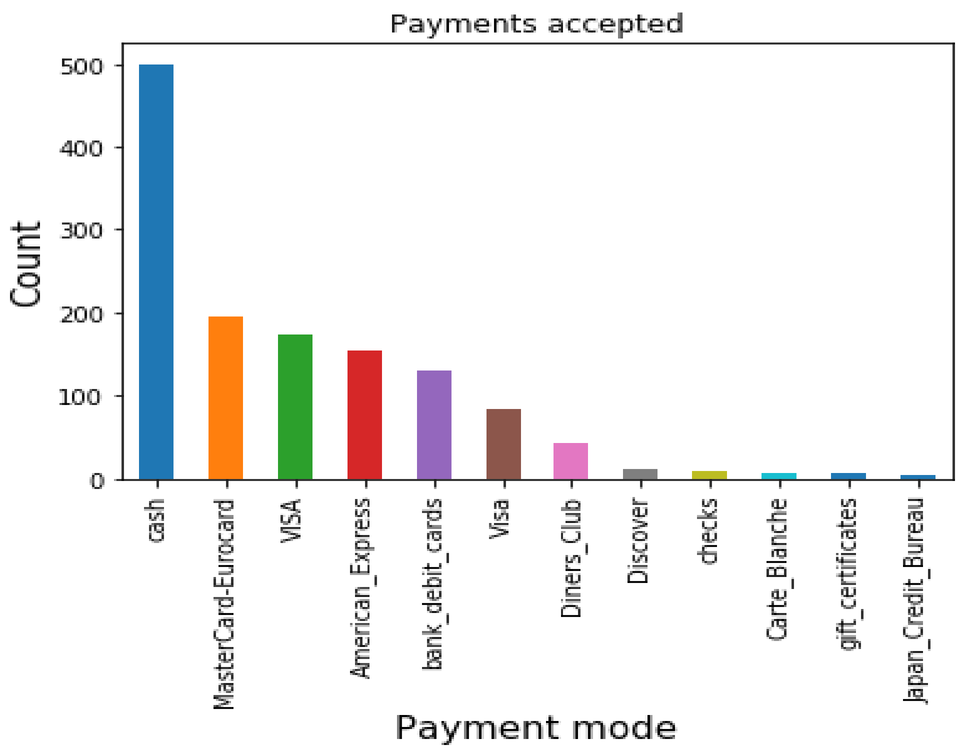


Figure 7.4 Graph depicting the most accepted payment mode

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

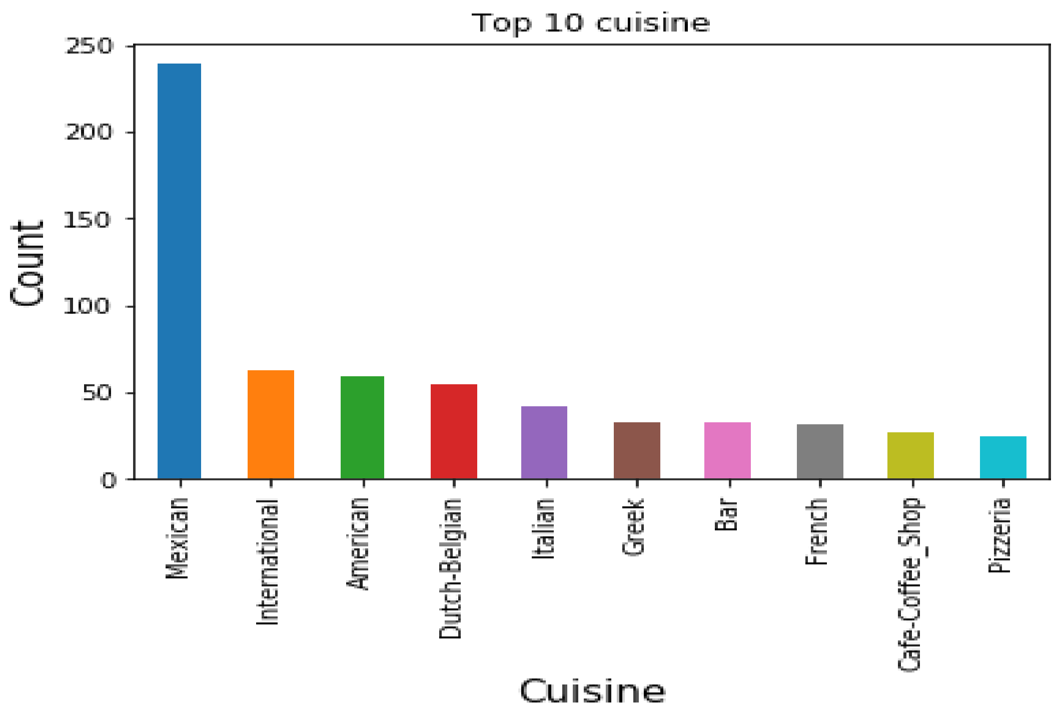


Figure 7.5 Graph depicting the top cuisines

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.

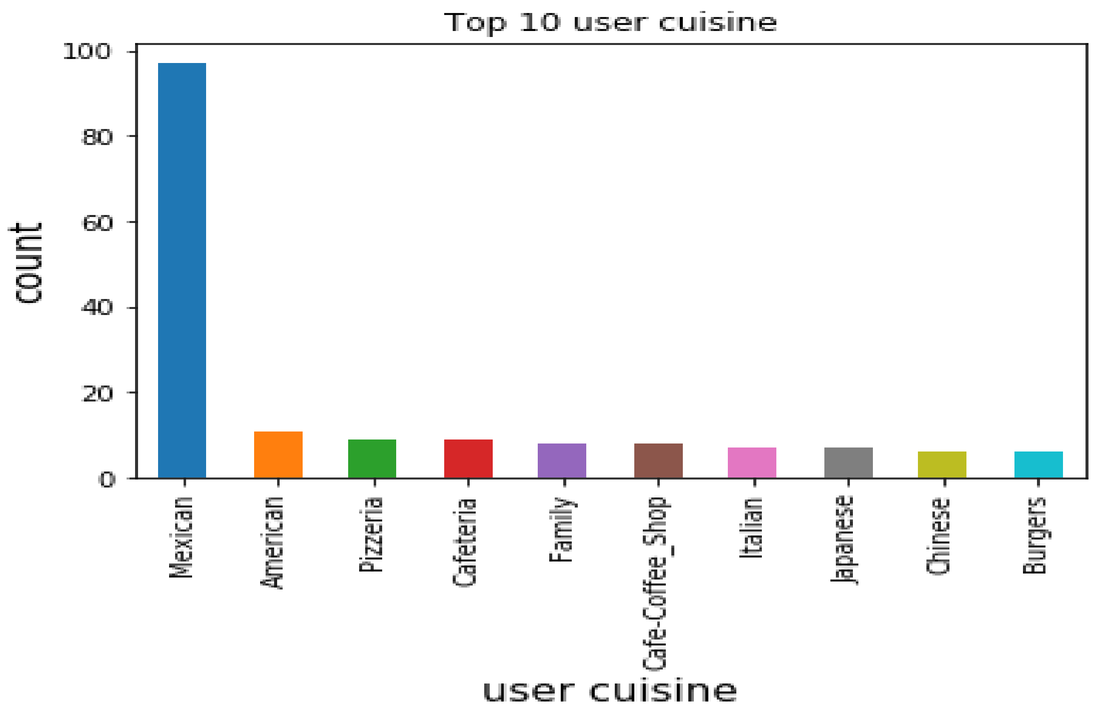


Figure 7.6 Graph depicting the top favourite cuisines for customers

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.

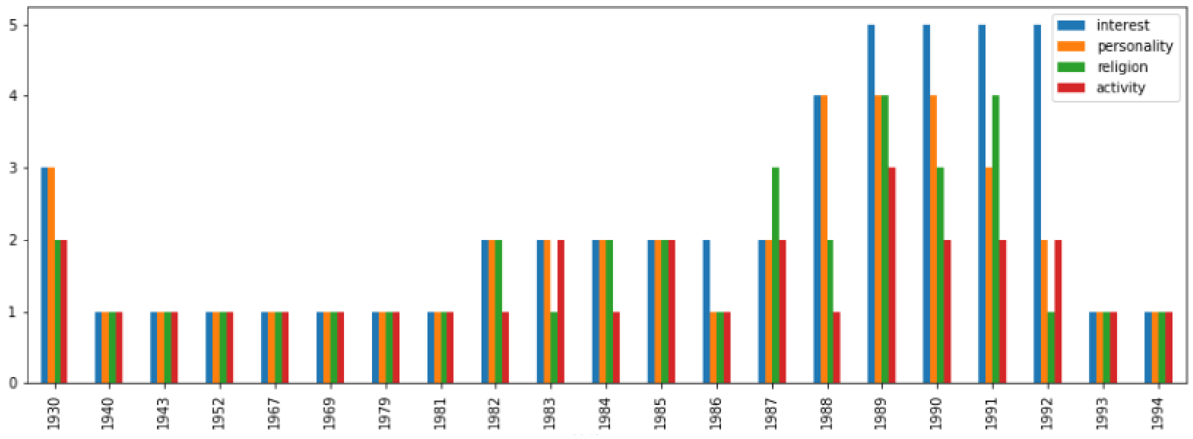


Figure 7.7 Graph depicting user's personal information based on birth year

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.