National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Muhammad Imran Shaikh
Student ID: x17119308

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Muhammad Imran Shaikh
Student ID:	x17119308
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	1458
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	25th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Muhammad Imran Shaikh
x17119308

1 Introduction

The main reason to write this configuration manual is to demonstrate the configuration of system setup, software, and hardware compatibility to run and implement the programming language code which will help to design the Research project and report. This manual will cover sections like System Configuration, Project Development, Codes Implementation, and Experiments with different machine learning models.

2 System Configuration

2.1 Hardware

Processor:3rd Generation Intel Core i5-3320M (2.6 GHz, 3MB L3 cache, 2 cores)1 Up to 3.30 GHz, Ram:16gb,System type:64-bit OS, Graphics: NVIDIA Quadro K2000M with 2GB dedicated DDR3, Windows:10pro(2019)

2.2 Software

Microsoft Office 356: Microsoft Word (For all the professional written stuff), Microsoft Excel(For storing dataset as CSV and Excel format and for visualization purpose too), Microsoft PowerPoint(presentation slides).

Python coding Language: Loading Libraries, Data Cleaning, Data preprocessing and engineering, Initial Data Analysis, Training and Test data splitting, Models Implementation, Hyperparameter Tuning, and Evaluation.Python IDE: Jupyter Notebook and PyCharm.

3 Project Development

The main steps involved in the project development phase are the selection for ideal IDE(Jupyter Notebook) to perform our coding task, Loading suitable libraries, Data cleaning(checking null values and imputations with aggregations), and Data preprocessing and engineering(grouping and joining datasets, Data merging, Describing columns, Removal of unnecessary features, Removal of pipes with string split into Genre columns).Initial data visualization(Word Cloud, several bar charts).

Preparation of separate class for movie dataset to implement Recommendation system techniques(Content-based filtering and Collaborative filtering), Splitting train and test set to fit in various models of our recommendation engine. Getting Top-N movies results from

our Recommendation machine learning models by utilizing different techniques. Models Hyperparameter tuning (for extracting the best parameters for optimal results). Models Evaluation(K-fold cross-validation,LOO(Leave One Out) Cross-validation with splits to get better Accuracy results from our machine learning models)Multiple Evaluation plots are plotted to get visual analysis.

3.1 Data Extraction and Pre-processing

The dataset is collected and generated by GroupLens Research Group, Dataset (ml-latest-small)[1] consist of almost 100k ratings by different users with over 1200 movie tags from 9125 movies. 'ml' stands for movie lens. Each selected user had at least rated 20 movies. There are 4 files included in this dataset named 'movies.csv', 'ratings.csv', 'tags.csv', 'links.csv' but for recommendation purpose, we are considering only two dataset files i.e. 'movies.csv' and 'rating.csv' The entire coding is done in the python programming language. Various python Libraries are imported based on the implementation of different recommendation techniques. All Recommendation system models are imported from 'Surprise Library' which is an official python recommendation system library can be seen in Figure 1 . Data preprocessing and engineering are obtained by grouping and joining

Importing libraries

n [103]:

```
import io # Module for Python interfaces to stream handling
import os #  Module for using operating system dependent functionality
import csv # Reading csv files
from surprise import Reader # Reader Function
import sys # Module contains functions to allow python to interact with system
import pandas as pd # Pandas library
import numpy as np # Numpy library
import matplotlib.pyplot as plt # matplotlib visualization library
from surprise.model_selection import LeaveOneOut # LeaveOneOut Cross Validator
from surprise.model_selection import cross_validate #cross validation module in surprise lib
from surprise import KNNBaseline # KNN baseline algorithm
from surprise import KNNWithMeans # KNN with means Algorithm
from surprise import SVD,SVDpp # Singular Value decomposition and Singular Value decomposition ++
from surprise import Dataset
from surprise import accuracy # Accuracy function
import seaborn as sns # Seaborn library for visulaization
from surprise.model_selection import KFold # K-fold cross validator
from surprise import get_dataset_dir # Extracting dataset from folder directory
from functools import reduce # function that implements a mathematical technique of folding or reduction
```

Figure 1: Importing python libraries

datasets, Data merging, Describing columns, deletion of unnecessary features, Removal of pipes with string split in Genre columns, Create a function that counts the number of times each genre appear can be seen in Figure 2.

4 Implementation of Recommendation Engine Machine Learning Models

Our recommendation engine is tested with 3 different recommendation system techniques i.e. (Content-based filtering, Collaborative filtering, and Matrix Factorization) by resulting Top-N movies results for users and even for movies in content-based filtering. A separate Movie class is generated to combine both movies and ratings CSV files by grouping them by users and movie ids numbers can be seen in the figure. Different machine

[1] `https://grouplens.org/datasets/movielens/latest/`

```
In [65]:
#define a function that counts the number of times each genre appear:
def count_word(df, ref_col, liste):
    keyword_count = dict()
    for s in liste: keyword_count[s] = 0
    for liste_keywords in df[ref_col].str.split('|'):
        if type(liste_keywords) == float and pd.isnull(liste_keywords): continue
        for s in liste_keywords:
            if pd.notnull(s): keyword_count[s] += 1
    # convert the dictionary in a list to sort the keywords  by frequency
    keyword_occurences = []
    for k,v in keyword_count.items():
        keyword_occurences.append([k,v])
    keyword_occurences.sort(key = lambda x:x[1], reverse = True)
    return keyword_occurences, keyword_count

In [66]:
#here we  make census of the genres:
genre_labels = set()
for s in data['genres'].str.split('|').values:
    genre_labels = genre_labels.union(set(s))

In [67]:
#counting how many times each of genres occur:
keyword_occurences, dum = count_word(data, 'genres', genre_labels)
keyword_occurences

Out[67]: [['Drama', 4365],
          ['Comedy', 3315],
          ['Thriller', 1729],
          ['Romance', 1545],
          ['Action', 1545],
          ['Adventure', 1117],
          ['Crime', 1100],
          ['Horror', 877],
          ['Sci-Fi', 792],
          ['Fantasy', 654],
          ['Children', 583],
          ['Mystery', 543],
          ['Documentary', 495],
          ['Animation', 447],
          ['Musical', 394],
          ['War', 367],
          ['Western', 168],
          ['IMAX', 153],
          ['Film-Noir', 133],
          ['(no genres listed)', 18]]
```

Figure 2: Create a function that counts the number of times each genre appears

learning models are implemented with their defined parameters to get better recommendations. Surprise library which is an official python recommendation system library has been utilized to implement machine learning models for our recommendation engine Figure 1.Following are the steps that are taken to create and evaluate our recommendation engine.

4.1 Experiment with Movielens Dataset Analysis

The Analysis on movielens dataset based on the merging of two dataset files i.e. 'movies.csv' and 'rating.csv' as an inner join by movies Ids in fig. With the help of this merged dataset, we are visualizing movie genres by the creation of world cloud and histogram to analyses which movie genres are most popular can be seen in Figure 3, Figure 4. Moreover, top 25 movies with the highest ratings are also plotted to analyze which movie is rated highest by different users in Figure 5.

4.2 Experiment with Content Based Filtering

Content-based filtering works on the phenomena of users' interest in different items. So, based on that interest, similar items are recommended to the users. Recommendation action becomes more accurate if the user provides more input. In our Content-based recommendation engine, we are finding 10 nearest neighbors of the movie of our interest by implementing the KNNBaseline algorithm with a similarity metric of Pearson baseline Figure 6.

In [68]:

```
# Function that control the color of the words
def random_color_func(word=None, font_size=None, position=None,
                      orientation=None, font_path=None, random_state=None):
    h = int(360.0 * tone / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(70, 120)) / 255.0)
    return "hsl({}, {}%, {}%)".format(h, s, l)


#Finally, the result is shown as a wordcloud:
words = dict()
trunc_occurences = keyword_occurences[0:50]
for s in trunc_occurences:
    words[s[0]] = s[1]
tone = 100 # define the color of the words
f, ax = plt.subplots(figsize=(14, 6))
wordcloud = WordCloud(width=550,height=300, background_color='black',
                      max_words=1628,relative_scaling=0.7,
                      color_func = random_color_func,
                      normalize_plurals=False)
wordcloud.generate_from_frequencies(words)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

Figure 3: Word cloud to analyze which Genre are the popular ones

```
In [71]:
# lets display the same result in the histogram
fig = plt.figure(1, figsize=(18,13))
ax2 = fig.add_subplot(2,1,2)
y_axis = [i[1] for i in trunc_occurences]
x_axis = [k for k,i in enumerate(trunc_occurences)]
x_label = [i[0] for i in trunc_occurences]
plt.xticks(rotation=85, fontsize = 15)
plt.yticks(fontsize = 15)
plt.xticks(x_axis, x_label)
plt.ylabel("No. of occurences", fontsize = 24, labelpad = 0)
ax2.bar(x_axis, y_axis, align = 'center', color='b')
plt.title("Popularity of Genres",bbox={'facecolor':'k', 'pad':5},color='B',fontsize = 30)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\pylabtools.py:128: MatplotlibDeprecationWarning: Support for uppercase
single-letter colors is deprecated since Matplotlib 3.1 and will be removed in 3.3; please use lowercase instead.
  fig.canvas.print_figure(bytes_io, **kw)
```

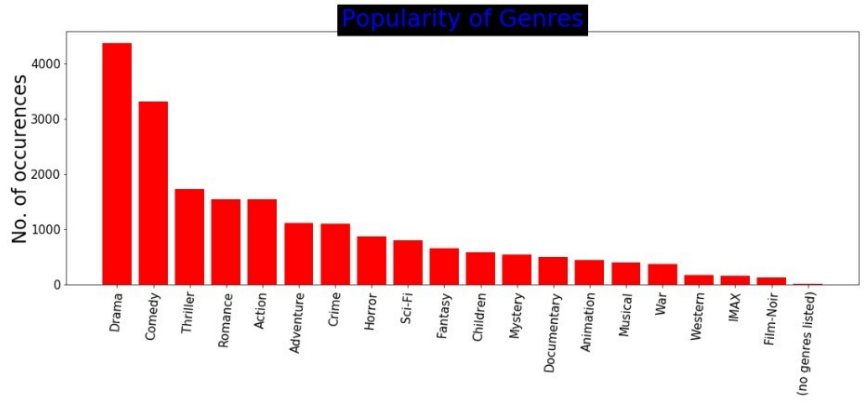

Figure 4: Histogram to analyze which Genre are the popular ones

```
In [70]:
plt.figure(figsize=(30,10))
plt.title('Top 25 movies with highest rating',fontsize=40)
colors=['red','yellow','orange','green','magenta','cyan','blue','lightgreen','skyblue','purple']
plt.ylabel('ratings',fontsize=30)
plt.xticks(fontsize=25,rotation=90)
plt.xlabel('movies title',fontsize=30)
plt.yticks(fontsize=25)
sns.barplot(x=high_rated.index,y=high_rated.rating,palette='rainbow')
```

```
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x20747827080>
```

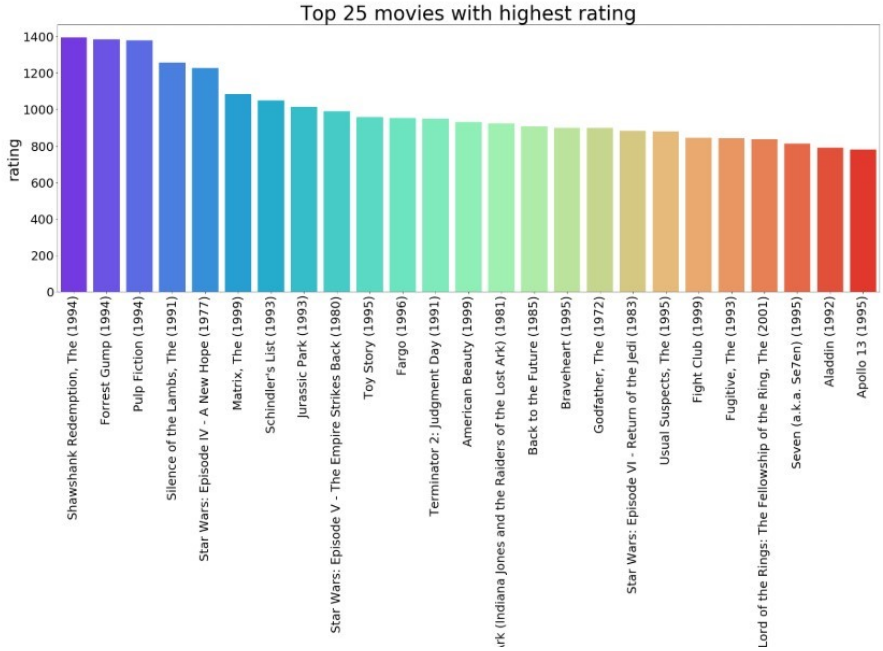

Figure 5: Top 25 movies with highest ratings

Using pearson_baseline as a simliarity measure in KnnBasic algorithm

In [20]:
```
sim_options = {'name': 'pearson_baseline', 'user_based': False}
algo = KNNBaseline(k=30 ,sim_options=sim_options)
algo.fit(trainset)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

Out[20]:
```
<surprise.prediction_algorithms.knns.KNNBaseline at 0x1b7fe2666d8>
```

In [21]:
```
# Read the mappings raw id into movie name
rid_to_name, name_to_rid = read_item_names()
```

In [22]:
```
# Fetching inner id of the movie

movie_raw_id = name_to_rid['Clockwork Orange, A (1971)']
movie_inner_id = algo.trainset.to_inner_iid(movie_raw_id)
movie_neighbors = algo.get_neighbors(movie_inner_id, k=10)

# Convert inner ids of the neighbors into names.
movie_neighbors = (algo.trainset.to_raw_iid(inner_id)
                   for inner_id in movie_neighbors)
movie_neighbors = (rid_to_name[rid]
                   for rid in movie_neighbors)
```

Output

In [23]:
```
print()
print('The 10 nearest neighbors of {} are:'.format(rid_to_name[movie_raw_id]))
for movie in movie_neighbors:
    print(movie)
```

```
The 10 nearest neighbors of Clockwork Orange, A (1971) are:
Apocalypse Now (1979)
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)
GoodFellas (1990)
Crumb (1994)
This Is Spinal Tap (1984)
People vs. Larry Flynt, The (1996)
Full Metal Jacket (1987)
Godfather, The (1972)
Pulp Fiction (1994)
Psycho (1960)
```

Figure 6: Content based filtering technique by using KNNBaseLine algorithm

4.3 Experiment with User and Item(Memory) Based Collaborative Filtering

User and Item-based collaborative filtering are one of the most extensively used techniques in the recommendation system, it works by finding a group of similar users who have given the similar reactions to the item of your interest. The rating matrix is created to find similar users and items based on ratings that are given by the user.KNNwithMean machine learning algorithm along with similarity metric of Cosine is utilized to get Top-10 nearest neighbor movies for specific user Figure 7, Figure 8.

User based Collaborative Filtering

In [187]:
```
trainset, testset = train_test_split(data, test_size=.25)
sim_options = {'name': 'cosine','user_based': True}
sim_options_other = {'name': 'pearson','user_based': True}
```

Initiating our KNNWithMeans Model and compute the similarities

In [188]:
```
model = KNNWithMean(k=50, sim_options=sim_options)
model.fit(trainSet)
simsMatrix = model.compute_similarities()
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Get top N similar users to our test subject

In [189]:
```
User = '85' # User id defined here
k = 10 # Top-10 Neighbours
testUserInnerID = trainSet.to_inner_uid(User)
similarityRow = simsMatrix[testUserInnerID]

similarUsers = []
for innerID, score in enumerate(similarityRow):
    if (innerID != testUserInnerID):
        similarUsers.append( (innerID, score) )
kNeighbors = heapq.nlargest(k, similarUsers, key=lambda t: t[1])
```

Get the stuff they rated, and add up ratings for each item, weighted by user similarity

In [190]:
```
candidates = defaultdict(float)
for similarUser in kNeighbors:
    innerID = similarUser[0]
    userSimilarityScore = similarUser[1]
    theirRatings = trainSet.ur[innerID]
    for rating in theirRatings:
        candidates[rating[0]] += (rating[1] / 5.0) * userSimilarityScore
```

Create a dictionary of stuff the user has already seen

In [191]:
```
watched = {}
for itemID, rating in trainSet.ur[testUserInnerID]:
    watched[itemID] = 1
```

Output

In [192]:
```
# Get top-rated items from similar users:
print("\nUser based Collaborative filtering results for user: {}\n".format(User))
pos = 0
for itemID, ratingSum in sorted(candidates.items(), key=itemgetter(1), reverse=True):
    if not itemID in watched:
        movieID = trainSet.to_raw_iid(itemID)

        print(ml.getMovieName(int(movieID)))
        pos += 1
        if (pos > 10):
            break
```

```
User based Collaborative filtering results for user: 85
```

Figure 7: User based collaborative filtering technique by using KNNWithMean algorithm

Item based Collaborative Filtering

Switching user_based parameter in (sim_options_icf) to (False) to utilize item based collaborative filtering

In [129]:
```
sim_options_icf = {'name': 'cosine','user_based': False}
sim_options_icf_other = {'name': 'pearson','user_based': False}
```

Intiating our KNNBasic Algorithm and fit our model in trainSet

In [130]:
```
model_icf = KNNWithMeans(sim_options=sim_options_icf)
model_icf.fit(trainSet)
simsMatrix_icf = model_icf.compute_similarities()
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Top N Items

In [131]:
```
Item = '85'
k = 10
testItemInnerID = trainSet.to_inner_uid(Item)

# Get the top K items we rated
testItemRatings = trainSet.ur[testItemInnerID]
kNeighbors_icf = heapq.nlargest(k, testItemRatings, key=lambda t: t[1])
```

Candidates Generation

In [132]:
```
# Get similar items to stuff we liked (weighted by rating)
candidates_icf = defaultdict(float)
for itemID_icf, rating_icf in kNeighbors_icf:
    similarityRow_icf = simsMatrix_icf[itemID_icf]
    for innerID_icf, score_icf in enumerate(similarityRow_icf):
        candidates_icf[innerID_icf] += score_icf * (rating_icf / 5.0)
```

Already watched item dictionary

In [133]:
```
watched_items = {}
for itemID_icf, rating_icf in trainSet.ur[testItemInnerID]:
    watched_items[itemID_icf] = 1
```

Item based Output

In [134]:
```
print("\nItem based Collaborative filtering results for user: {}\n".format(Item))
# Get top-rated items from similar users:
pos_icf = 0
for itemID_icf, ratingSum_icf in sorted(candidates_icf.items(), key=itemgetter(1), reverse=True):
    if not itemID_icf in watched_items:
        movieID = trainSet.to_raw_iid(itemID_icf)
        print(ml.getMovieName(int(movieID)))
        pos_icf += 1
        if (pos_icf > 10):
            break
```

```
Item based Collaborative filtering results for user: 85

James Dean Story, The (1957)
Get Real (1998)
```

Figure 8: Item based collaborative filtering technique by using KNNWithMean algorithm

4.4 Experiment with Matrix Factorization(Model)Based Collaborative Filtering

In Matrix factorization or Model-based collaborative filtering is the Dimensionality Reduction technique just like Principal Component Analysis(PCA). Matrix factorization breaks down the user-item large matrix into a smaller matrix. The hidden features are defined by latent factors that are created by item and user column and row matrix. In our Matrix factorization method, we are implementing two matrix factorization algorithms i.e. SVD(Singular Value Decomposition) and SVDpp(Singular Value Decomposition plus plus) to get Top-10 recommendation results shown in Figure 9, Figure 10.

Matrix Factorization

In [230]:
```
# using above defined trainedset
trainset = trainset
def GetAntiTestSetForUser(testSubject):
        fill = trainset.global_mean
        anti_testset = []
        u = trainset.to_inner_uid(str(testSubject))
        user_items = set([j for (j, _) in trainset.ur[u]])
        anti_testset += [(trainset.to_raw_uid(u), trainset.to_raw_iid(i), fill) for
                                i in trainset.all_items() if
                                i not in user_items]
        return anti_testset
```

In [231]:
```
# Defining svd model and fitting training set

model_svd = SVD()
model_svd.fit(trainset)
User = 85
testset = GetAntiTestSetForUser(User)
predictions_svd = model_svd.test(testset)
```

In [232]:
```
recommendations_svd = []
print("\nMatrix factorization results using SVD for user: {}\n".format(User))
for userID, movieID, actualRating, estimatedRating, _ in predictions_svd:
    intMovieID = int(movieID)
    recommendations_svd.append((intMovieID, estimatedRating))
    recommendations_svd.sort(key=lambda x: x[1], reverse=True)

for ratings in recommendations_svd[:10]:
    print(ml.getMovieName(ratings[0]))
```

```
Matrix factorization results using SVD for user: 85

Eternal Sunshine of the Spotless Mind (2004)
Matrix, The (1999)
Dark Knight, The (2008)
Maltese Falcon, The (1941)
Wallace & Gromit: A Close Shave (1995)
Her (2013)
Godfather, The (1972)
Lock, Stock & Two Smoking Barrels (1998)
Indiana Jones and the Last Crusade (1989)
Roger & Me (1989)
```

Figure 9: Matrix Factorization technique by using SVD algorithm

SVD++ Algorithm

```
In [224]:
model_svdpp = SVDpp()
model_svdpp.fit(trainset)
testset = GetAntiTestSetForUser(85)
predictions_SVDpp = model_svdpp.test(testset)
```

```
In [225]:
recommendations_svdpp = []
print("\nMatrix factorization results using SVDpp for user: {}\n".format(User))
for userID, movieID, actualRating, estimatedRating, _ in predictions_SVDpp:
    intMovieID = int(movieID)
    recommendations_svdpp.append((intMovieID, estimatedRating))
    recommendations_svdpp.sort(key=lambda x: x[1], reverse=True)

for ratings in recommendations_svdpp[:10]:
    print(ml.getMovieName(ratings[0]), ratings[1])
```

```
Lock, Stock & Two Smoking Barrels (1998) 4.791976290151504
Star Wars: Episode VI - Return of the Jedi (1983) 4.683258519259071
Lord of the Rings: The Fellowship of the Ring, The (2001) 4.589518342901994
It Happened One Night (1934) 4.587213958522119
Grand Day Out with Wallace and Gromit, A (1989) 4.582666195126913
Matrix, The (1999) 4.554760912512864
Moon (2009) 4.519638374812435
Lone Star (1996) 4.46500649106575
Lord of the Rings: The Two Towers, The (2002) 4.457753030447094
12 Angry Men (1957) 4.44088989608321
```

Figure 10: Matrix Factorization technique by using SVD++ algorithm

5 Experiment with Models Hyperparameter Tuning

For Models Hyperparameter tuning we are considering Grid Search CV from Python Surprise Library, which provides us the best parameters to get optimal value from our machine learning models when we are training our dataset. The main parameters which are considered for KNN and Matrix factorization-based algorithms are different K-Values, Epoches no, learning rate, Similarity options, and accuracy measures(RMSE and MAE) can be seen in Figure 11, Figure 12.

User-based Collaborative Filtering

```
In [34]:
param_grid_KWM_UB = {'k': [10, 20, 50], 'n_epochs': [1, 3, 5],
                     'learning_rate': [.00001,.00003,.00005],
              'sim_options': {'name': ['msd', 'cosine'],
                              'min_support': [1, 5],
                              'user_based': [True]}
                }
gs_KWM_UB = GridSearchCV(KNNWithMeans, param_grid_KWM_UB, measures=['rmse', 'mae'], cv=3)
gs_KWM_UB.fit(data)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine 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...
```

```
In [35]:
# We can now use the algorithm that yields the best rmse:
algo_KWM_UB = gs_KWM_UB.best_estimator['rmse']  # pass the best model to algo
print(gs_KWM_UB.best_score['rmse'])
# combination of parameters that gave the best RMSE score
print(gs_KWM_UB.best_params['rmse'])
cross_validate(algo_KWM_UB, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
0.9269129465163269
{'k': 50, 'n_epochs': 1, 'learning_rate': 1e-05, 'sim_options': {'name': 'msd', 'min_support': 1, 'user_based': True}}
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
```

Figure 11: Hyperparameter tuning with GridSearchCV for different params of collaborative filtering models

```
# ----- SVD ----- #

param_grid = {'n_factors': [110, 120, 140, 160], 'n_epochs': [90, 100, 110], 'lr_all': [0.001, 0.003, 0.005, 0.008],
              'reg_all': [0.08, 0.1, 0.15]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)
algo = gs.best_estimator['rmse']
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Use the new parameters with the train data
algo = SVD(n_factors=160, n_epochs=100, lr_all=0.005, reg_all=0.1)
algo.fit(trainset)
test_pred = algo.test(testset)
print("SVD : Test Set")
accuracy.rmse(test_pred, verbose=True)

# ----- SVD++ ----- #

param_grid = {'n_factors': [20, 30, 40], 'n_epochs': [20,30,40], 'lr_all': [0.001, 0.003, 0.005, 0.008],
              'reg_all': [0.08, 0.1, 0.15]}
gs = GridSearchCV(SVDpp, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)
algo = gs.best_estimator['rmse']
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Use the new parameters with the train data
algo = SVDpp(n_factors=40, n_epochs=40, lr_all=0.008, reg_all=0.1)
algo = SVDpp()
algo.fit(trainset)
test_pred = algo.test(testset)
print("SVD++ : Test Set")
accuracy.rmse(test_pred, verbose=True)
```

Figure 12: Hyperparameter tuning with GridSearchCV for different params of Matrix Factorization models

6 Experiment with Models Evaluation

For Models Cross-Validation, we are considering two cross validators to test the accuracy of our recommendation engine models in different splits. Those two Cross validators are K-Fold cross Validator and LOOCV leave one out cross validator can be seen in Figure 13, Figure 14

K-Fold Cross Validator

In [31]:
```
# define a cross-validation iterator

kf = KFold(n_splits=3)

# Defining models

# KNN Baseline Algo for Content based Recommendation
sim_option_KNN = {'name': 'pearson_baseline', 'user_based': False}
algo_KNN_CB  = KNNBaseline(sim_options=sim_option_KNN)

# KNN with mean Algo for User based Collaborative Filtering
sim_options = {'name': 'cosine','user_based': True}
algo_KWM_UB = KNNWithMeans(k=50, sim_options=sim_options)

# KNN with mean Algo for Item based Collaborative Filtering
sim_options_IB = {'name': 'cosine','user_based': False}
algo_KWM_IB = KNNWithMeans(k=50, sim_options=sim_options_IB)

# SVD(Singular Value Decomposition) model for Matrix Factorization
algo_svd = SVD()

# SVDpp(Singular Value Decomposition)plus plus model for Matrix Factorization
algo_svdpp = SVDpp()

# Dictionary for folds accuracy
rmse_KNN = []
mae_KNN = []
rmse_KWM_UB = []
mae_KWM_UB = []
rmse_KWM_IB = []
mae_KWM_IB = []
rmse_svd = []
mae_svd = []
rmse_svdpp = []
mae_svdpp = []

# Training and testing data on basis of K-Folds
for trainset, testset in kf.split(data):

    # Fitting Train and test sets in models for predictions

    # KNN Baseline Algo for Content based Recommendation
    algo_KNN_CB.fit(trainset)
    predictions_KNN = algo_KNN_CB.test(testset)

    # KNN with mean Algo for User based Collaborative Filtering
    algo_KWM_UB.fit(trainset)
    predictions_KWM_UB = algo_KWM_UB.test(testset)

    # KNN with mean Algo for Item based Collaborative Filtering
    algo_KWM_IB.fit(trainset)
    predictions_KWM_IB = algo_KWM_IB.test(testset)

    # SVD(Singular Value Decomposition) model for Matrix Factorization
    algo_svd.fit(trainset)
    predictions_svd = algo_svd.test(testset)

    # SVDpp(Singular Value Decomposition)plus plus model for Matrix Factorization
    algo_svdpp.fit(trainset)
    predictions_svdpp = algo_svdpp.test(testset)
```

Figure 13: Models Evaluation with K-fold Cross Validation

Leave One Out Cross Validator

In [38]:
```
# define a Leave one out cross-validation iterator

Looc = LeaveOneOut(n_splits=3)

# Defining models

# KNN BaseLine Algo for Content based Recommendation
sim_option_KNN = {'name': 'pearson_baseline', 'user_based': False}
algo_KNN_CB  = KNNBaseline(sim_options=sim_option_KNN)

# KNN with mean Algo for User based Collaborative Filtering
sim_options = {'name': 'cosine','user_based': True}
algo_KWM_UB = KNNWithMeans(k=50, sim_options=sim_options)

# KNN with mean Algo for Item based Collaborative Filtering
sim_options_IB = {'name': 'cosine','user_based': False}
algo_KWM_IB = KNNWithMeans(k=50, sim_options=sim_options_IB)

# SVD(Singular Value Decomposition) model for Matrix Factorization
algo_svd = SVD()

# SVDpp(Singular Value Decomposition)plus plus model for Matrix Factorization
algo_svdpp = SVDpp()

# Dictionary for folds Of LOOCV accuracy
rmse_KNN_L= []
mae_KNN_L = []
rmse_KWM_UB_L = []
mae_KWM_UB_L = []
rmse_KWM_IB_L = []
mae_KWM_IB_L = []
rmse_svd_L = []
mae_svd_L = []
rmse_svdpp_L = []
mae_svdpp_L = []

# Training and testing data on basis of Leave One Out CV
for trainset, testset in Looc.split(data):

    # Fitting Train and test sets in models for predictions

    # KNN Baseline Algo for Content based Recommendation
    algo_KNN_CB.fit(trainset)
    predictions_KNN = algo_KNN_CB.test(testset)

    # KNN with mean Algo for User based Collaborative Filtering
    algo_KWM_UB.fit(trainset)
    predictions_KWM_UB = algo_KWM_UB.test(testset)

    # KNN with mean Algo for Item based Collaborative Filtering
    algo_KWM_IB.fit(trainset)
    predictions_KWM_IB = algo_KWM_IB.test(testset)

    # SVD(Singular Value Decomposition) model for Matrix Factorization
    algo_svd.fit(trainset)
    predictions_svd = algo_svd.test(testset)

    # SVDpp(Singular Value Decomposition)plus plus model for Matrix Factorization
    algo_svdpp.fit(trainset)
    predictions_svdpp = algo_svdpp.test(testset)
```

Figure 14: Models Evaluation with LOO(Leave One Out) Cross Validation