RECOMENDATION TOOLS INDIVIDUAL ASSIGNMENT

BY: NITHESH RAMANNA

Contents

INTRODUCTION

Recommendation systems is a crucial tool now a days for any e-commerce company, as it is going to suggest the products to the customers, which in turn acts as a virtual seller and may have an impact on increase in sales. This system also increases the user experience and makes shopping enjoyable.

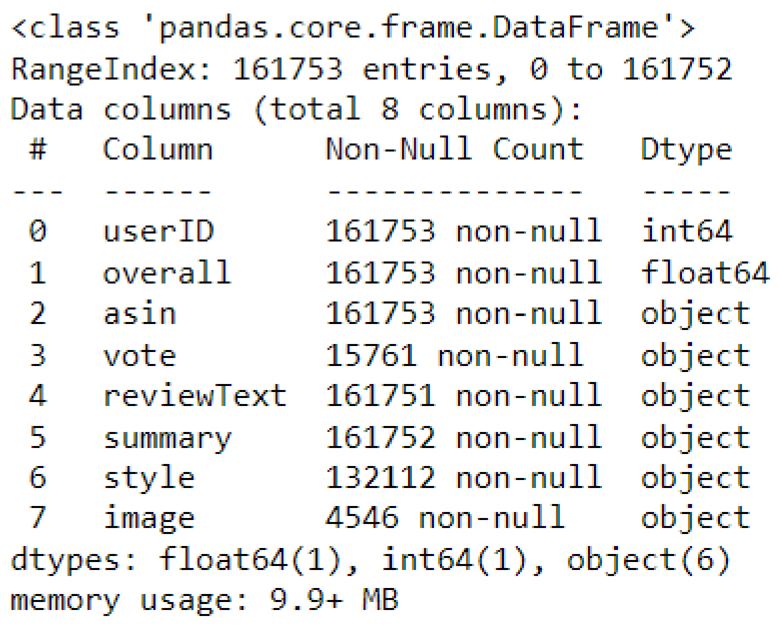
Recommendation systems are not only used in e-commerce but also used in other channels that we use in day-to-day life. Recommendation systems are applied today in most of the services like social media, OTT platforms, app stores and many more.

Recommendation system is a tool that uses algorithms, data analysis and machine learning to give the recommendations online. These kinds of recommendations may be particular to the customer or may be generic recommendations. The recommendations provided may be based on the data available and the algorithm used. For personalization, the system uses the data which is related to the user's profile and the kinds of searches made by the user and the ratings given to the products to recommend the relevant products to the user.

DATA

3 .csv dataset were there, out of which 2 were user review and item details from amazon and one test data to get the predictions.

train.csv file: There are 161753 entries in this dataset



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161753 entries, 0 to 161752
Data columns (total 8 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   userID      161753 non-null  int64
 1   overall     161753 non-null  float64
 2   asin        161753 non-null  object
 3   vote        15761 non-null   object
 4   reviewText  161751 non-null  object
 5   summary     161752 non-null  object
 6   style       132112 non-null  object
 7   image       4546 non-null    object
dtypes: float64(1), int64(1), object(6)
memory usage: 9.9+ MB
```

Variable Name	Type	Description
userID	Int	User Identification Number
overall	Float	Ratings ranges from (1 to 5)
asin	Object	Item Identification Number
vote	Object	
reviewText	Object	Text review
summary	Object	Review summary
style	Object	Item style (size, model etc)
Image	Object	Link of the image posted by user

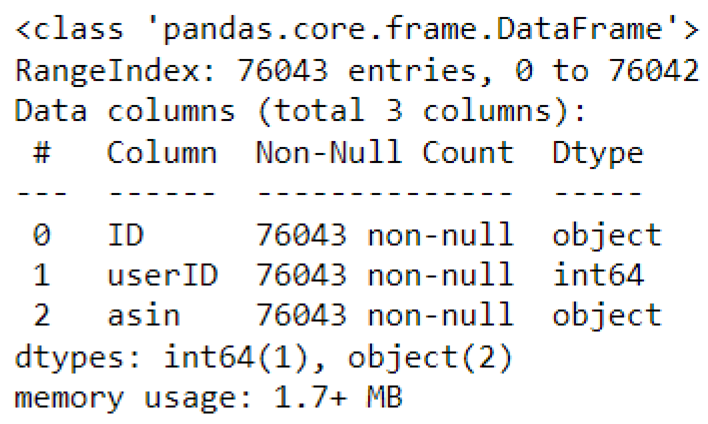
metadata.csv file: There are 2577 entries in this dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2577 entries, 0 to 2576
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   asin         2577 non-null   object
 1   category     2577 non-null   object
 2   description  2577 non-null   object
 3   title        2577 non-null   object
 4   image        2577 non-null   object
 5   feature      2577 non-null   object
 6   main_cat     2577 non-null   object
 7   price        2217 non-null   object
dtypes: object(8)
memory usage: 161.2+ KB
```

Variable Name	Type	Description
asin	Object	Item identification number
category	Object	Item category
description	Object	Item description
title	Object	Item title
image	Object	Item image
feature	Object	Item features

main_cat	Object	Main category the Item belongs
price	Object	Price of the Item

test_students.csv file: There were 76043 entries in the dataset.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76043 entries, 0 to 76042
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   ID      76043 non-null  object
 1   userID  76043 non-null  int64 
 2   asin    76043 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
```

Variable Name	**Type**	**Description**
ID	Object	Unique Identification by concatenating userID with asin
userID	Int	User identification number
asin	Object	Item Identification number

ALGORITHMS APPLIED

5 different algorithms have been applied on amazon review dataset for predicting the user rating. They are

1. User based collaborative filtering
2. Item based collaborative filtering
3. Matrix Factorization: SVD
4. Co-Clustering
5. Content Based recommendations

PRE-PROCESSING DATA

train.csv data has been subset with only 3 columns *[userID, overall, asin]* and split into train and test data with the test size equal to 30% of the whole data.

For content based, metadata.csv data has been used. 270 duplicate items have been removed from the metadata.

Surprise package class method load_from_df is used to build the train set for feeding the recommendation algorithms. Test data set has been converted to tuples.

ALGORITHMS EXPLANATION

USER BASED COLLABORATIVE FILTERING

Collaborative filtering is one of the techniques in recommendation systems which can be used to filter the items that any user like based on the likes or purchases made by similar kind of users.

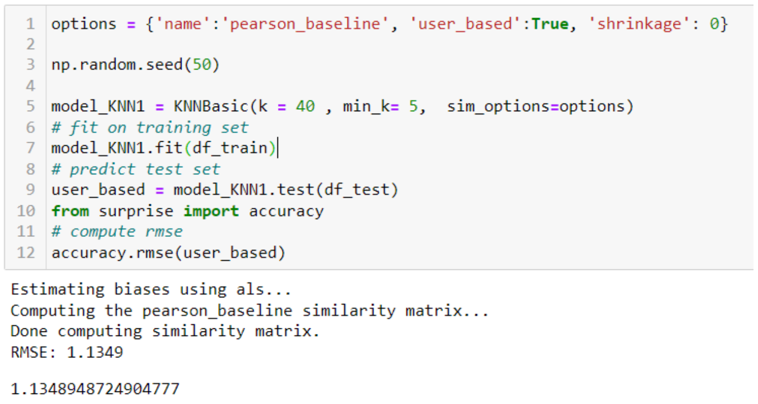
It just does this job by searching the large group of users and find the smaller groups of users with similar tastes. It creates the ranked list based on the items liked by users and give that list as suggestions for the users belong the group.

This recommendation algorithm needs the set of items and set of users who reacted to some of the items in the set. It can explicit like by rating on a scale of 1 to 5 or implicit like viewing the item or adding the item to their wish list. Here to evaluate the algorithm ratings on the scale of 1 to 5 have been used.

Collaborative filter algorithms work on 2 basic assumptions. One is users will give ratings to the items and other is users with similar interests in past have similar interests in future.

One way of collaborative filtering algorithm is user-based. The algorithm will work on the basic principle that given a user and an item which the user has not seen it at all in past how likely the user is going to like or rate the given item. This can be achieved by finding similar set of users who liked the items liked by the users we selected and take the average ratings of the similar users.

This algorithm is applied using KNNBasic method from surprise package and the similarity is measured by the Pearson correlation. Have used the parameters *k = 40 , min_k= 5* and *sim_options* with options *{'name':'pearson_baseline', 'user_based':True, 'shrinkage': 0}.*



```
options = {'name':'pearson_baseline', 'user_based':True, 'shrinkage': 0}

np.random.seed(50)

model_KNN1 = KNNBasic(k = 40 , min_k= 5,  sim_options=options)
# fit on training set
model_KNN1.fit(df_train)
# predict test set
user_based = model_KNN1.test(df_test)
from surprise import accuracy
# compute rmse
accuracy.rmse(user_based)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
RMSE: 1.1349

1.1348948724904777
```

Figure 1

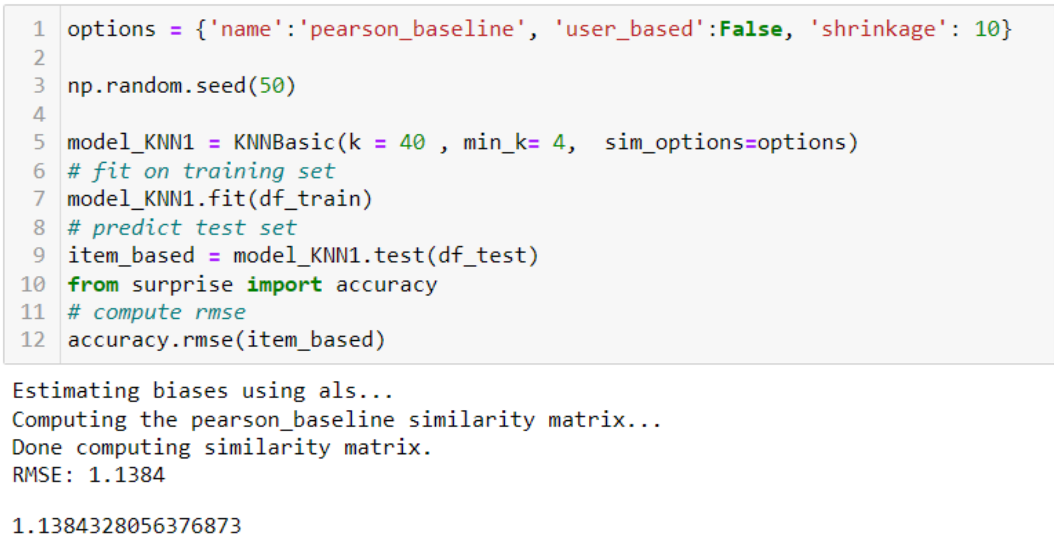
The rmse score for this approach can be seen from figure 1 and is 1.1349, which is high when compared to the benchmark set in the Kaggle competition.

Cons of user-based approach are scalability, space and time complexity and sparsity.

ITEM-BASED COLLABORATIVE FILTERING

This one of the other approaches of collaborative filtering. Unlike user based here the similarity is calculated based on the items not the users. The approach takes the items which are similar to the item we selected. For example, predicting the user's rating for a pair of shoes. The rating is predicted based on user's ratings for similar items.

This algorithm is applied using KNNBasic method from surprise package and the similarity is measured by the Pearson correlation. Have used the parameters *k = 40 , min_k= 4* and *sim_options* with options *{'name':'pearson_baseline', 'user_based':False, 'shrinkage': 10}.*



```
options = {'name':'pearson_baseline', 'user_based':False, 'shrinkage': 10}

np.random.seed(50)

model_KNN1 = KNNBasic(k = 40 , min_k= 4,  sim_options=options)
# fit on training set
model_KNN1.fit(df_train)
# predict test set
item_based = model_KNN1.test(df_test)
from surprise import accuracy
# compute rmse
accuracy.rmse(item_based)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
RMSE: 1.1384

1.1384328056376873
```

Figure 2

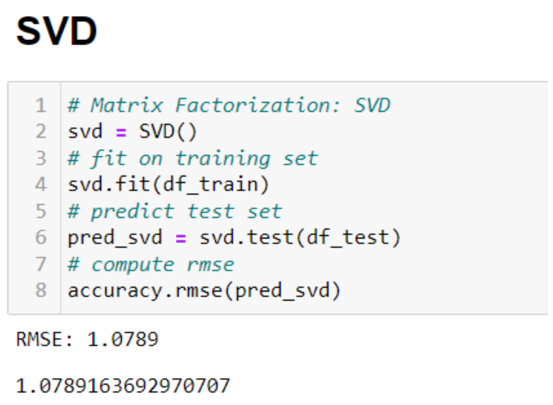
Looking at figure 2, both user-based and item-based approaches performed similarly. The rmse score here is 1.1384.

Cons of the item-based approach are Scalability problem like user based and memory requirements

MATRIX FACTORIZATION: SVD

SVD is a matrix factorization technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). In the context of the recommender system, the SVD is used as one of the collaborative filtering techniques. It uses a matrix structure where each row represents a user, and each column represents an item. The value in each cell of the matrix is the ratings.

The algorithm is applied using the SVD method from surprise package and it gave the rmse score of 1.0789 which is much better than user-based and item-based approach.



SVD

```
# Matrix Factorization: SVD
svd = SVD()
# fit on training set
svd.fit(df_train)
# predict test set
pred_svd = svd.test(df_test)
# compute rmse
accuracy.rmse(pred_svd)
```

```
RMSE: 1.0789

1.0789163692970707
```

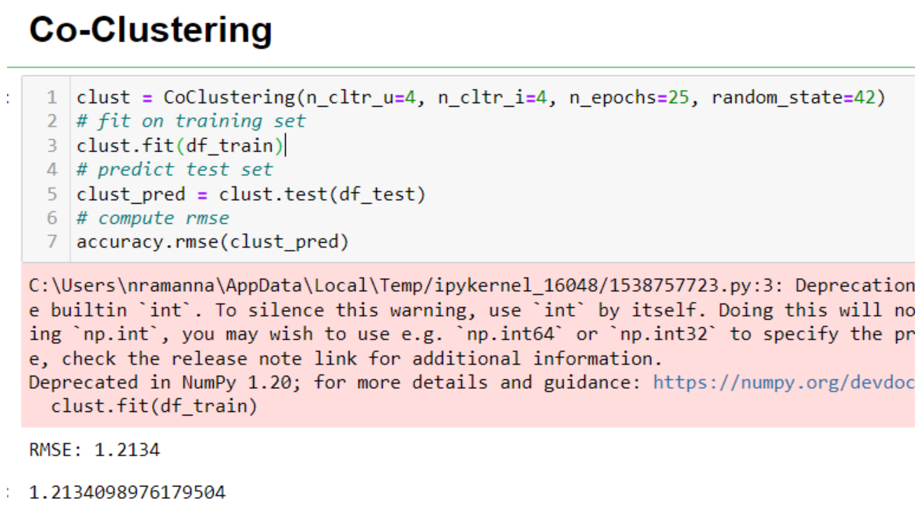
Figure 3

Pros of this algorithm are it simplifies data, removes noise and may improve algorithm results.

CO-CLUSTERING

Cluster analysis is done on the user item matrix to find similarity matrix. Here the users and the items are assigned to some user clusters and item clusters and some co-clusters. The cluster centers assigned is taken as predicted ratings. Basic idea of co-clustering is that predicting the simultaneous cluster for both users and items based on pairwise interaction similarity.

The algorithm is applied using the CoCLustering method from surprise package and it gave the rmse score of 1.2134. It can be said that this algorithm didn't do well. The parameters used is *n_cltr_u(number of clusters for users), n_cltr_i(number of clusters for items), n_epochs(number of iteration of optimization loop) and random_state.* Parameter values, *n_cltr_u = 4, n_cltr_i=4, n_epochs = 25, random_state = 42.*



Co-Clustering

```
clust = CoClustering(n_cltr_u=4, n_cltr_i=4, n_epochs=25, random_state=42)
# fit on training set
clust.fit(df_train)
# predict test set
clust_pred = clust.test(df_test)
# compute rmse
accuracy.rmse(clust_pred)
```

```
C:\Users\nramanna\AppData\Local\Temp/ipykernel_16048/1538757723.py:3: Deprecation
e builtin `int`. To silence this warning, use `int` by itself. Doing this will no
ing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the pr
e, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdoc
  clust.fit(df_train)
```

```
RMSE: 1.2134

1.2134098976179504
```

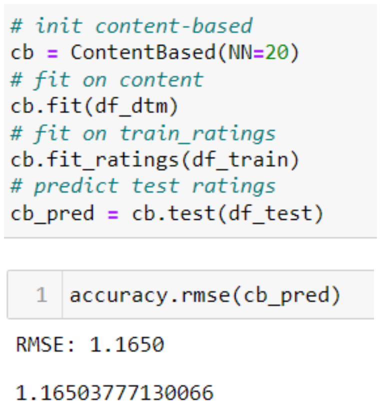
Figure 4

CONTENT-BASED RECOMMENDATIONS

Content based recommendation system works with the data provided by users implicitly or explicitly. The data generated by user profiles is used to make the suggestions to user. The more actions by users more accurate the results. The concepts of Term frequency and Inverse document frequency are used in content-based recommendation system. These concepts are used to determine the relative importance of the article.

For content based, metadata of the item is used. The description of the item is tokenized. The stop words are removed and stemmed the words to get the root forms of the words. The tokenized words are used to create the TFIDF matrix using the *TfidfVectorizer()* method from *sklearn.feature_extraction.text* package.

The *ContentBased* class given is used to fit the mode. The TFIDF matrix is used to fit the contents to the model. To fit the ratings to model train data, which is mentioned in the preprocessing step is used.



```
# init content-based
cb = ContentBased(NN=20)
# fit on content
cb.fit(df_dtm)
# fit on train_ratings
cb.fit_ratings(df_train)
# predict test ratings
cb_pred = cb.test(df_test)
```

```
accuracy.rmse(cb_pred)
```

```
RMSE: 1.1650

1.16503777130066
```

Figure 5

The rmse for content-based is 1.1650.

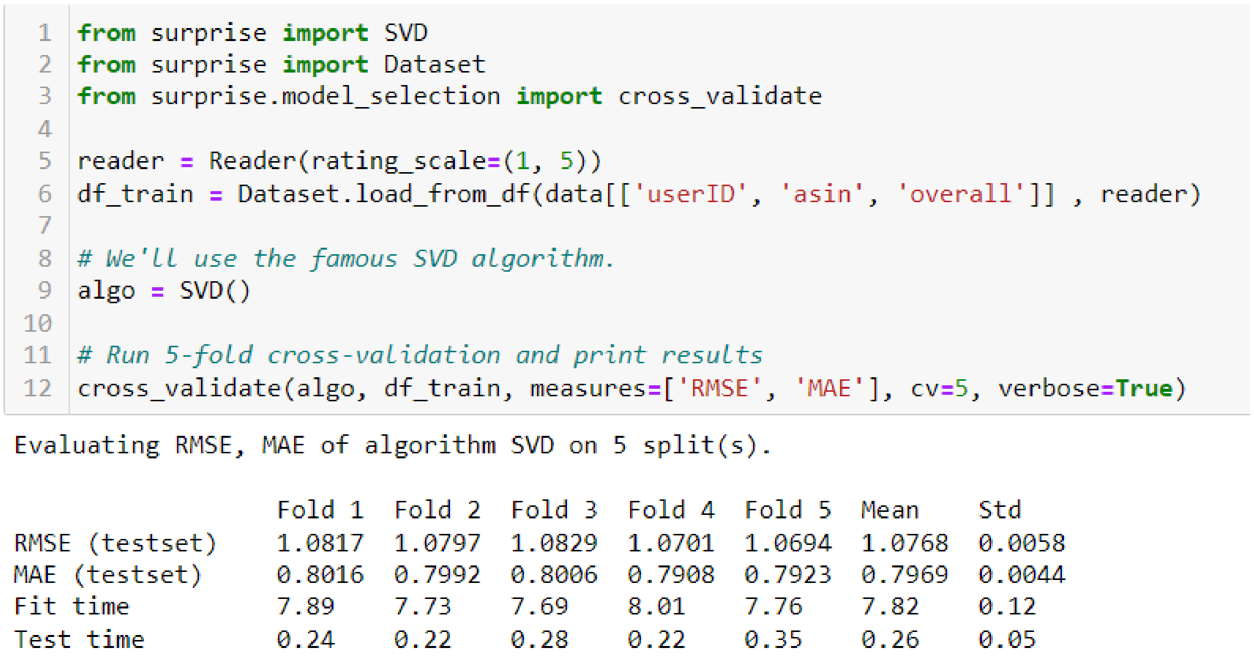
Pros of this model are it doesn't require the user data; this makes it easy to scale to many users and it captures the specific interest of the users and gives the recommendations of items to users even though very few users showed interest in those items.

Cons of this model are, it requires more domain knowledge, and the model can make recommendations based on users' existing interests.

As seen results of all the algorithms, SVD gave the best result so far. The rmse is **1.0789** which is lesser than any algorithms rmse scores. So, I have decided to tune hyper parameters of the SVD method to get the best predictions

TUNING HYPER PARAMETERS

Before trying to tune the parameter, cross validation is conducted with 5-fold, in which rmse of all the folds gave the same results which were around 1.07.



```python
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

reader = Reader(rating_scale=(1, 5))
df_train = Dataset.load_from_df(data[['userID', 'asin', 'overall']] , reader)

# We'll use the famous SVD algorithm.
algo = SVD()

# Run 5-fold cross-validation and print results
cross_validate(algo, df_train, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0817  1.0797  1.0829  1.0701  1.0694  1.0768  0.0058
MAE (testset)     0.8016  0.7992  0.8006  0.7908  0.7923  0.7969  0.0044
Fit time          7.89    7.73    7.69    8.01    7.76    7.82    0.12
Test time         0.24    0.22    0.28    0.22    0.35    0.26    0.05
```

Looking at the results tried to make my first submission with this results in Kaggle. The results rmse in Kaggle gone up to 1.09714 which means I overfitted the model. It was not clear for me why did that happened initially as I have not used any parameters to get the best results. After 4 submissions I figured out that I was type casting the userID and asin to string while predicting the ratings.

Later, I ran the SVD algorithm without type casting, and made the 5th submission. Now the rmse score went down to 1.03972.

I did decide to choose the parameters *n_factors, n_epochs, lr_all, reg_all* for the best results. Before giving the combination of all the 4 parameters, 5-fold cross validation is conducted with each chosen parameter to get the best rmse score.

n_factors: number of factors – n_factors = 26 gave the least rmse

n_epochs: number of iterations – n_epochs = 16 gave the least rmse.

lr_all: learning rate for all parameters – lr_all = 0.007 gave the least rmse.

reg_all: regularization term for all parameters – reg_all = 0.2 gave the least rmse.

Initially using these parameter values ratings were predicted and submissions were made which just crossed the benchmark. I just made the combination of these parameter slightly increasing the *n_factors* and *n_epochs,* which gave the good results when compared to the benchmark in Kaggle. I make around 5-6 submission like that.

Later have conducted the 10-fold cross validation with grid search using GridSearchCV method from surprise package. For my 13th submission in Kaggle I have used the grid search.

With *param_grid = {'n_epochs': [40,50, 60], 'lr_all': [0.006,0.007,0.008],'reg_all': [0.2, 0.3, 0.4], 'n_factors':[90,100,110]}.*This gave the rmse score of 1.061 in personal system and rmse score of 1.00120 in Kaggle. The best values for parameters are *{'n_epochs':60, 'lr_all': 0.008,'reg_all': 0.2, 'n_factors':110}.*

It can be seen that the higher values of *n_epochs*, *lr_all* and *n_factors.* So, gird search was conducted with 10-fold cross validation with *param_grid = {'n_epochs': [60,70,80], 'lr_all': [0.008,0.009],'reg_all': [0.2, 0.3, 0.4], 'n_factors':[110,120,130]}.* This combination gave the rmse score of 1.056 and 0.99288 in the Kaggle, this was my 14th submission. The best values for parameters are *{'n_epochs':80, 'lr_all': 0.009,'reg_all': 0.2, 'n_factors':130}.*

For 15th submission, gird search was conducted with 10-fold cross validation with *param_grid = {'n_epochs': [80,90,100], 'lr_all': [0.008,0.009],'reg_all': [0.2, 0.3, 0.4], 'n_factors':[130,140,150]}.* This combination gave the rmse score of 1.055 and 0.98982 in the Kaggle submission. The best values for parameters are *{'n_epochs':100, 'lr_all': 0.009,'reg_all': 0.2, 'n_factors':150}.*

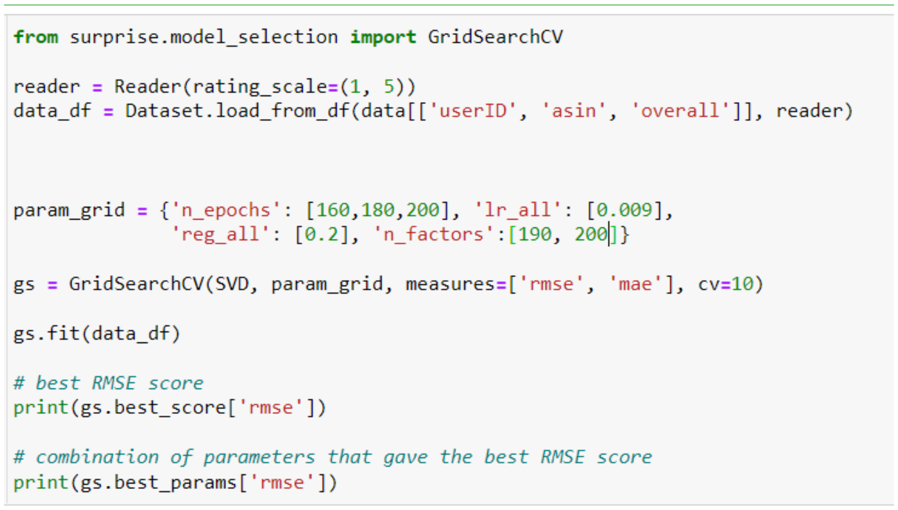
For 16th submission, gird search was conducted with 10-fold cross validation with *param_grid = {'n_epochs': [100,110,120], 'lr_all': [0.009],'reg_all': [0.2], 'n_factors':[150,160,170]}.* This combination gave the rmse score of 1.054 and 0.98680 in the Kaggle submission. The best values for parameters are *{'n_epochs':120, 'lr_all': 0.009,'reg_all': 0.2, 'n_factors':170}.*

For 17th submission, gird search was conducted with 10-fold cross validation with *param_grid = {'n_epochs': [120,140,160], 'lr_all': [0.009],'reg_all': [0.2], 'n_factors':[170,190,210]}.* This combination gave the rmse score of 1.053 and 0.98361 in the Kaggle submission. The best values for parameters are *{'n_epochs':160, 'lr_all': 0.009,'reg_all': 0.2, 'n_factors':210}.*

For 18th submission, gird search was conducted with 10-fold cross validation with *param_grid = {'n_epochs': [160,180,200], 'lr_all': [0.009],'reg_all': [0.2], 'n_factors':{190,200]}.* This combination gave the rmse score of 1.052 and 0.98406 in the Kaggle submission. The best values for parameters are *{'n_epochs':200, 'lr_all': 0.009,'reg_all': 0.2, 'n_factors':200}.* Though this grid-search cross validation gave lesser rsme in the system, it slightly increased the rmse score in Kaggle when compared to submission 17th.

I stopped at 18th submission as the rmse got increased with the grid search. So, the best predication in Kaggle was on 17th submission with rmse score of **0.98361**.

Below is the code for grid search



```python
from surprise.model_selection import GridSearchCV

reader = Reader(rating_scale=(1, 5))
data_df = Dataset.load_from_df(data[['userID', 'asin', 'overall']], reader)



param_grid = {'n_epochs': [160,180,200], 'lr_all': [0.009],
              'reg_all': [0.2], 'n_factors':[190, 200]}

gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=10)

gs.fit(data_df)

# best RMSE score
print(gs.best_score['rmse'])

# combination of parameters that gave the best RMSE score
print(gs.best_params['rmse'])
```

Figure 6

REFERENCES

1. Professors jupyter notebooks
2. https://surprise.readthedocs.io/en/stable/ - surprise package documentation