13.06.1443

RECOMMENDATION SYSTEM WITH SPARK

BIG DATA FINAL HOMEWORK REPORT

MUHAMMET FURKAN YILMAZ
181805069

ANIL DURMAZ
181805080

FERHAT KAMALI
171805056

Dataset Information

This dataset is large enough to build good recommendation model and is adapted from 'Netflix prize dataset' which is very large and you may face memory issue while training a model using that dataset.

Netflix held the Netflix Prize open competition for the best algorithm to predict user ratings for films.

Movie File Description; Movie File Contains Movie_ID, Name, Year

Rating File Description; Rating File Contains Movie*ID, User*ID, Rating
Rating : 1 - 5

Movie_ID	Year	Name
1.00 - 889.45 Count: 889 1 17.8k	1915 2005	17297 unique values
1	2003	Dinosaur Planet
2	2004	Isle of Man TT 2004 Review
3	1997	Character
4	1994	Paula Abdul's Get Up & Dance
5	2004	The Rise and Fall of ECW
6	1997	Sick
7	1992	8 Man

User_ID	Rating	Movie_ID
6 2.65m	1 5	3 4496
712664	5	3
1331154	4	3
2632461	3	3
44937	5	3
656399	4	3
439011	1	3
1644750	3	3

1. Firstly we use pandas library for reading two 'csv' dataset and merge them.

```
In [3]:     # reading two 'csv' dataset and merge them
        df_rating = pd.read_csv(r'Netflix_Dataset_Rating\Netflix_Dataset_Rating.csv')
        df_movie = pd.read_csv(r'Netflix_Dataset_Rating\Netflix_Dataset_Movie.csv')
        df = pd.merge(df_movie, df_rating,how='right', on=["Movie_ID"] )
```

2. After that we got that database;

	Movie_ID	Year	Name	User_ID	Rating
0	3	1997	Character	712664	5
1	3	1997	Character	1331154	4
2	3	1997	Character	2632461	3
3	3	1997	Character	44937	5
4	3	1997	Character	656399	4
...	...	...	...	...	...
17337453	4496	1993	Farewell My Concubine	520675	3
17337454	4496	1993	Farewell My Concubine	1055714	5
17337455	4496	1993	Farewell My Concubine	2643029	4
17337456	4496	1993	Farewell My Concubine	1559566	3
17337457	4496	1993	Farewell My Concubine	293198	3

17337458 rows × 5 columns

3. We take random 50000 rows inside of that;

```
In [5]: netfix_df = df.sample(n = 50000, replace=True, random_state=33)

In [6]: netfix_df = netfix_df.sort_index()

In [7]: netfix_df
```

Out[7]:

	Movie_ID	Year	Name	User_ID	Rating
745	3	1997	Character	2382844	5
1001	3	1997	Character	1689050	4
1160	3	1997	Character	288727	3
1780	8	2004	What the #$*! Do We Know!?	1084149	3
2063	8	2004	What the #$*! Do We Know!?	1863874	5
...	...	...	...	...	...
17336379	4496	1993	Farewell My Concubine	158607	4
17336386	4496	1993	Farewell My Concubine	2110064	5
17336494	4496	1993	Farewell My Concubine	1995067	5
17336913	4496	1993	Farewell My Concubine	1908051	3
17337319	4496	1993	Farewell My Concubine	1426829	3

50000 rows × 5 columns

4. We printed new database info;

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Movie_ID  50000 non-null  int64 
 1   Year      50000 non-null  int64 
 2   Name      50000 non-null  object
 3   User_ID   50000 non-null  int64 
 4   Rating    50000 non-null  int64 
dtypes: int64(4), object(1)
memory usage: 1.9+ MB
None
```

5. We delete punctuation from movie names;

```
In [10]: import re

         chars_to_remove = "#$&*!.,;?_-:()\/'"
         rx = re.escape(''.join(chars_to_remove))
         print(rx)
         i = 0
         for row in netfix_df['Name']:
             netfix_df['Name'][i] = re.sub('[%s]' % re.escape(chars_to_remove), '', netfix_df['Name'][i])
             print(i,"  ",netfix_df['Name'][i])
             i = i+1
```

```
\#\$\&\*!\.,;\?_\-:\(\)\\/'
0    Character
1    Character
2    Character
3    What the  Do We Know
4    What the  Do We Know
5    What the  Do We Know
6    What the  Do We Know
7    What the  Do We Know
8    What the  Do We Know
9    What the  Do We Know
10    What the  Do We Know
11    What the  Do We Know
12    What the  Do We Know
13    What the  Do We Know
14    What the  Do We Know
15    What the  Do We Know
16    What the  Do We Know
17    What the  Do We Know
```

6. We verify here is there any empty variable in new database;

```
In [11]: netfix_df.to_csv('FAF.csv', index=False)

In [12]: netfix_df[netfix_df.isna()==True].count()

Out[12]: Movie_ID    0
         Year        0
         Name        0
         User_ID     0
         Rating      0
         dtype: int64

In [13]: netfix_df.isna()

Out[13]:
```

	Movie_ID	Year	Name	User_ID	Rating
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
49995	False	False	False	False	False
49996	False	False	False	False	False
49997	False	False	False	False	False
49998	False	False	False	False	False
49999	False	False	False	False	False

50000 rows × 5 columns

8. We get dataset tos park like this;

```
pyspark_df = spark.read.csv('FAF.csv', header=True)
```

```
pyspark_df=pyspark_df.withColumn("Year",pyspark_df.Year.cast("int"))
pyspark_df=pyspark_df.withColumn("User_ID",pyspark_df.User_ID.cast("int"))
pyspark_df=pyspark_df.withColumn("Movie_ID",pyspark_df.Movie_ID.cast("int"))
pyspark_df=pyspark_df.withColumn("Rating",pyspark_df.Rating.cast("int"))
```

9. We check dataframe in here;

```
In [21]: print("Type of Netfix Dataframe : ",type(pyspark_df))
         print("Shape of Netfix Dataframe : ",pyspark_df.toPandas().shape)

         Type of Netfix Dataframe :  <class 'pyspark.sql.dataframe.DataFrame'>
         Shape of Netfix Dataframe :  (50000, 5)
```

```
In [22]: pyspark_df.groupBy("Movie_ID").count().show()

         +--------+-----+
         |Movie_ID|count|
         +--------+-----+
         |     148|   65|
         |     463|    7|
         |     471|   17|
         |     833|   27|
         |    1238|    6|
         |    1645|   48|
         |    1959|   14|
         |    2122|  215|
         |    2366|    6|
         |    2659|    4|
         |    2866|   47|
         |     897|   28|
         |    1395|   22|
         |    1507|    6|
         |    1721|   13|
         |    2235|    8|
         |    2580|  150|
         |    3226|   31|
         |    3475|    7|
         |    4161|    1|
         +--------+-----+
         only showing top 20 rows
```

10. Create ALS Model for every parameters;

```
model_parameters = []
rank = [10,50,200]
iteration = [10,50,200]
lamb = [0.01,0.1]
als_models = []
for r in rank:
    for i in iteration:
        for l in lamb:
            model_parameters.append("rank:" + " " + str(r) + " " + "||  Iteration:" + " " + str(i) + " " + "||  Lambda:" + " " +
            als = ALS(rank = r, maxIter=i, regParam=l,userCol='User_ID', itemCol='Movie_ID', ratingCol ='Rating',
                      seed = 5080, coldStartStrategy="drop")
            als_models.append(als)
```

11. Firstly we tried get the code in to the for loop for try different als models but we encountered a problem in for loop and we try to make that with hand. But ve encontered problem again and we could not fix the problem;

```
In [30]: als_models[17]
Out[30]: ALS_bc4a8e69d8f4

In [31]: fitted_models = []
         import time

In [32]: fitted_models.append(als_models[0].fit(train))
         time.sleep(5)

In [33]: fitted_models.append(als_models[1].fit(train))
         time.sleep(5)

In [34]: fitted_models.append(als_models[2].fit(train))
         time.sleep(5)

         ---------------------------------------------------------------------------
         Py4JJavaError                             Traceback (most recent call last)
         ~\AppData\Local\Temp/ipykernel_10988/1187853997.py in <module>
         ----> 1 fitted_models.append(als_models[2].fit(train))
               2 time.sleep(5)

         C:\spark\python\pyspark\ml\base.py in fit(self, dataset, params)
             130                 return self.copy(params)._fit(dataset)
             131             else:
         --> 132                 return self._fit(dataset)
             133         else:
             134             raise ValueError("Params must be either a param map or a list/tuple of param maps, "

         C:\spark\python\pyspark\ml\wrapper.py in _fit(self, dataset)
             293
             294     def _fit(self, dataset):
         --> 295         java_model = self._fit_java(dataset)
```

12. We calculated Root Mean Square Error for every learned model;

```
for i in range(len(fitted_models)):
    print("Model's Parameters: ")
    print(model_parameters[i])
    minSquare(fitted_models[i])

Model's Parameters:
rank: 10 ||  Iteration: 10 ||  Lambda: 0.01
Root-mean-square error = 4.056312699840548
```

13. We generate top ten movie recommendations for each user and each movie;

```
# Generate top 10 movie recommendations for each user
userRecs = fitted_models[0].recommendForAllUsers(10)
# Generate top 10 user recommendations for each movie
movieRecs = fitted_models[0].recommendForAllItems(10)
```

```
sonuc = userRecs
result = sonuc.toPandas()
result['recommendations'][0]
```

```
[Row(Movie_ID=1972, rating=3.998771905899048),
 Row(Movie_ID=4011, rating=3.823249101638794),
 Row(Movie_ID=1089, rating=3.5958199501037598),
 Row(Movie_ID=3769, rating=3.512016773223877),
 Row(Movie_ID=433, rating=3.496905565261841),
 Row(Movie_ID=1790, rating=3.411038875579834),
 Row(Movie_ID=722, rating=3.263188123703003),
 Row(Movie_ID=2921, rating=3.2116596698760986),
 Row(Movie_ID=599, rating=3.1180572509765625),
 Row(Movie_ID=2212, rating=3.1112396717071533)]
```

```
result = movieRecs.toPandas()
film_name = "7 Seconds"
df = pyspark_df.toPandas()
film_ID = df[df['Name']==film_name]['Movie_ID']
film_ID = film_ID.unique()[0]
```

```
sonuc = list(result[result['Movie_ID']==film_ID]['recommendations'])
```

```
sonuc
for i in range(len(sonuc[0])):
    print(f"User {i+1}: ",sonuc[0][i][0])
```

```
User 1:  510262
User 2:  973219
User 3:  775189
User 4:  2475007
User 5:  1888914
User 6:  2311741
User 7:  2029979
User 8:  2589899
User 9:  638822
User 10:  330549
```

14. In second part we used Cosinus Similarity for recommendation system, first we took 5000 elements from dataset (because CountVectorizer requires too much ram);

```
df2 = df.sample(n = 5000, replace=True, random_state=33)
df2 = df2.sort_index()
index = pd.Index(range(0,5000,1))
df2 = df2.set_index(index)
df2
```

	Movie_ID	Year	Name	User_ID	Rating
0	3	1997	Character	2382844	5
1	16	1996	Screamers	1283299	4
2	17	2005	7 Seconds	420537	4
3	17	2005	7 Seconds	88661	4
4	18	1994	Immortal Beloved	2108751	4
...	...	...	...	...	...
4995	4496	1993	Farewell My Concubine	179647	4
4996	4496	1993	Farewell My Concubine	1670943	5
4997	4496	1993	Farewell My Concubine	1493191	2
4998	4496	1993	Farewell My Concubine	1001461	3
4999	4496	1993	Farewell My Concubine	1906611	4

5000 rows × 5 columns

15. Then we colected the important feautres together and saved them to new column;

```
def get_important_features(data):
    important_features = []
    for i in range(data.shape[0]):
        important_features.append(str(data['Year'][i]) + ' ' + str(data['Name'][i]) + ' ' + str(data['Rating'][i]))

    return important_features
```

```
df2['important_features'] = get_important_features(df2)
df2.head(3)
```

	Movie_ID	Year	Name	User_ID	Rating	important_features
0	3	1997	Character	2382844	5	1997 Character 5
1	16	1996	Screamers	1283299	4	1996 Screamers 4
2	17	2005	7 Seconds	420537	4	2005 7 Seconds 4

16. Then we Vectorized the new column and applied Cosinus similarity to this vector;

```
cm = CountVectorizer().fit_transform(df2['important_features'])
```

```
cs = cosine_similarity(cm)
print(cs)
```

```
[[1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]]
```

17. Lastly we sorted the results and printed them ;

```
sorted_scores = sorted(scores, key = lambda x:x[1], reverse = True)
sorted_scores = sorted_scores[1:]
print(sorted_scores)
```

```
movieTitle = []
title = "7 Seconds"
j = 0
print('The 10 most recomended movies to', title, 'are:\n')
for item in sorted_scores:
    movieTitle.append(df2[df2['index'] == item[0]]['Name'].values[0])
    j+=1
    if j > 90:
        break
result = unique2(movieTitle)

for i in range(len(result)):
    print(i+1, result[i])
```

```
The 10 most recomended movies to 7 Seconds are:

1 Daredevil
2 Elephant
3 Elf
4 Honey
5 Identity
6 Normal
7 SWAT
8 Somethings Gotta Give
9 Spun
10 Thirteen
```

Computer info

```python
import socket
hostname = socket.gethostname()
IPAddr = socket.gethostbyname(hostname)
print("Your Computer Name is:" + hostname)
print("Your Computer IP Address is:" + IPAddr)
```

```
Your Computer Name is:DESKTOP-BE0E817
Your Computer IP Address is:192.168.56.1
```

Task distribution

Ferhat and Furkan make first research of Project and find some example codes and different datasets. After that Anıl found a more usable dataset than the ones found by Furkan and Ferhat and we decided to use it.

Anıl was mainly responsible for ALS learning and preprocessing of the dataset. Merging the dataset, removing punctuations, removing the empty and repeating elements and lastly the ALS function is written by him.

Ferhat was mainly responsible for the CosineSimilarity function, taking a part from the dataset, collecting important features into one column, vectorization and cs learning algorithms are made by him.

Lastly Furkan was responsible for debugging and visualizations, repeatedly learning models, printing the results in human understandable way and fixing some errors are made by him.

Although Ferhat and Furkan also helped, most of the code was written by Anıl.