Movie Recommendation Engine

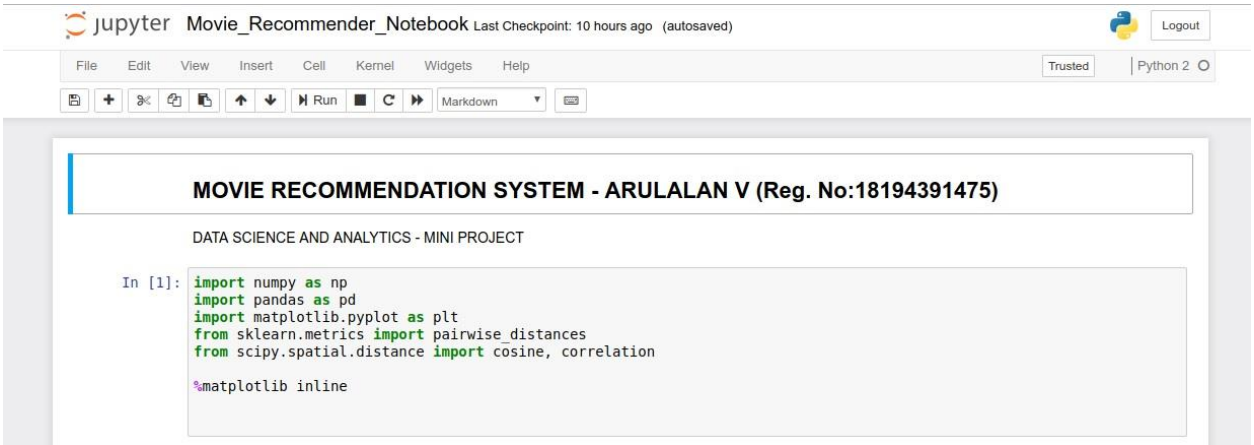
Recommender systems being a part of information filtering system are used to forecast the bias or ratings the user tend to give for an item. Among different kinds of recommendation approaches, collaborative filtering technique has a very high popularity because of their effectiveness. These traditional collaborative filtering systems can even work very effectively and can produce standard recommendations, even for wide ranging problems.

DATASET: 100k MovieLens Dataset https://grouplens.org/datasets/movielens/100k/

REQUIREMENT: Python-**Jupyter Notebook** with **Anaconda Package**

Steps for Implementation

Step 1: Importing packages



Step 2: Writing PIG script for choosing datasets

In [2]:
```
%%writefile extractdataset.pig

DEFINE preprocess1() returns data
{
        $data = load '/ml-100k/u.user' using PigStorage('|');
};

DEFINE preprocess2() returns data
{
        $data = load '/ml-100k/u.data' using PigStorage('|');
};

DEFINE preprocess3() returns data
{
        $data = load '/ml-100k/u.item' using PigStorage('|');
};


u = preprocess1();
rmf /prodata/processed_dataset/user/
store u into '/prodata/processed_dataset/user/' using PigStorage('|');

r = preprocess2();
rmf /prodata/processed_dataset/rating/
store r into '/prodata/processed_dataset/rating/' using PigStorage('|');

i = preprocess3();
rmf /prodata/processed_dataset/items/
store i into '/prodata/processed_dataset/items/' using PigStorage('|');
```

```
Overwriting extractdataset.pig
```

Step 3: Executing PIG script

```
In [3]: %%bash --err pig_out --bg
        pig -f extractdataset.pig

        Starting job # 0 in a separate thread.

In [4]: import sys;
        while True:
            line = pig_out.readline()
            if not line:
                break
            sys.stdout.write("%s" % line)
            sys.stdout.flush()
        Total bytes written : 19505491
        Spillable Memory Manager spill count : 0
        Total bags proactively spilled: 0
        Total records proactively spilled: 0

        Job DAG:
        job_local1648776587_0003


        2018-05-29 13:58:34,088 [main] INFO  org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with
         processName=JobTracker, sessionId= - already initialized
        2018-05-29 13:58:34,089 [main] INFO  org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with
         processName=JobTracker, sessionId= - already initialized
        2018-05-29 13:58:34,091 [main] INFO  org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with
         processName=JobTracker, sessionId= - already initialized

        2018-05-29 13:58:34,103 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunche
        r - Success!
        2018-05-29 13:58:34,241 [main] INFO  org.apache.pig.Main - Pig script completed in 48 seconds and 243 milliseconds
        (48243 ms)
```

Step 4: Loading datasets in **JUPYTER NOTEBOOK** and defining the columns

```
In [5]: from subprocess import Popen,PIPE
        p=Popen(['hdfs dfs -get /prodata/processed_dataset/rating/part-m-00000 /home/hduser/processed_dataset/r/'],shell=True
        stdout,stderr=p.communicate()
        print stdout,stderr
        p=Popen(['hdfs dfs -get /prodata/processed_dataset/user/part-m-00000 /home/hduser/processed_dataset/u/'],shell=True,s
        stdout,stderr=p.communicate()
        print stdout,stderr
        p=Popen(['hdfs dfs -get /prodata/processed_dataset/items/part-m-00000 /home/hduser/processed_dataset/i/'],shell=True,
        stdout,stderr=p.communicate()
        print stdout,stderr

        None
        None
        None

In [6]: u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
        users = pd.read_csv('/home/hduser/processed_dataset/u/part-m-00000', sep='|', names=u_cols,
                            encoding='latin-1', parse_dates=True)

        r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
        ratings = pd.read_csv('/home/hduser/processed_dataset/r/part-m-00000', sep='\t', names=r_cols,
                              encoding='latin-1')

        m_cols = ['movie_id', 'title', 'release_date', 'video_release_date', 'imdb_url']
        movies = pd.read_csv('/home/hduser/processed_dataset/i/part-m-00000', sep='|', names=m_cols, usecols=range(5),
                             encoding='latin-1')
```

Step 5: Merging movies dataset and rating dataset

```
In [7]: movie_ratings = pd.merge(movies, ratings)
        df = pd.merge(movie_ratings, users)

        df.head(6)
```

Out[7]:

	movie_id	title	release_date	video_release_date	imdb_url	user_id	rating	unix_timestamp	age	sex	occupation	zip_code
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	308	4	887736532	60	M	retired	95076
1	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	308	5	887737890	60	M	retired	95076
2	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	308	4	887739608	60	M	retired	95076
3	7	Twelve Monkeys (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Twelve%20Monk...	308	4	887738847	60	M	retired	95076
4	8	Babe (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Babe%20(1995)	308	5	887736696	60	M	retired	95076
5	9	Dead Man Walking	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Dead%20Man%20...	308	4	887737194	60	M	retired	95076

Step 6: Viewing user dataset

```
In [10]: users.head(6)
```

Out[10]:

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101

Step 7: Preprocessing dataset by dropping unwanted **COLUMNS**

Data Pre-Processing

```
In [11]: df.drop(df.columns[[3,4,7]], axis=1, inplace=True)
         ratings.drop( "unix_timestamp", inplace = True, axis = 1 )
         movies.drop(movies.columns[[3,4]], inplace = True, axis = 1 )
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 0 to 99999
Data columns (total 9 columns):
movie_id        100000 non-null int64
title           100000 non-null object
release_date    99991 non-null object
```

Step 8: Viewing user dataset after dropping unwanted COLUMN

```
In [13]: ratings.head(6)
```

Out[13]:

	user_id	movie_id	rating
0	196	242	3
1	186	302	3
2	22	377	1
3	244	51	2
4	166	346	1
5	298	474	4

Step 9: Grouping movies dataset and rating dataset, and calculated mean rating

Movie Ratings

```
In [18]: movie_stats = df.groupby('title').agg({'rating': [np.size, np.mean]})
         movie_stats.head(5)
```

Out[18]:

	rating	
	size	mean
title		
'Til There Was You (1997)	9	2.333333
1-900 (1994)	5	2.600000
101 Dalmatians (1996)	109	2.908257
12 Angry Men (1957)	125	4.344000
187 (1997)	41	3.024390

Step 10: Setting THRESHOD value

Setting a threshold of atleast 50 ratings for better analysis.

```
In [19]: min_50 = movie_stats['rating']['size'] >= 50
         movie_stats[min_50].sort_values([('rating', 'mean')], ascending=False).head()
```

Out[19]:

	rating	
	size	mean
title		
Close Shave, A (1995)	112	4.491071

Step 11: Creating PIVOT TABLE with user_id as column and movies as row

Pivot Table

```
In [22]: ratings_matrix = ratings.pivot_table(index=['movie_id'],columns=['user_id'],values='rating').reset_index(drop=True)
         ratings_matrix.fillna( 0, inplace = True )
         ratings_matrix.head(1684)
```

Out[22]:

user_id	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	941	942	943
0	5.0	4.0	0.0	0.0	4.0	4.0	0.0	0.0	0.0	4.0	...	2.0	3.0	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0
1	3.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1678	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1679	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1680	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1681	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1682 rows × 943 columns

Step 12: Calculating Cosine similarity for movies with user ratings and Distance Matrix is generated

Cosine Similarity

```
In [23]: movie_similarity = 1 - pairwise_distances( ratings_matrix.as_matrix(), metric="cosine" )
         np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use when sorting is done
         ratings_matrix = pd.DataFrame( movie_similarity )
         ratings_matrix.head(1684)
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	...	1672	1673	1674	1675	1
0	0.000000	0.402382	0.330245	0.454938	0.286714	0.116344	0.620979	0.481114	0.496288	0.273935	...	0.035387	0.000000	0.000000	0.000000	0.035
1	0.402382	0.000000	0.273069	0.502571	0.318836	0.083563	0.383403	0.337002	0.255252	0.171082	...	0.000000	0.000000	0.000000	0.000000	0.000
2	0.330245	0.273069	0.000000	0.324866	0.212957	0.106722	0.372921	0.200794	0.273669	0.158104	...	0.000000	0.000000	0.000000	0.000000	0.032
3	0.454938	0.502571	0.324866	0.000000	0.334239	0.090308	0.489283	0.490236	0.419044	0.252561	...	0.000000	0.000000	0.094022	0.094022	0.037
4	0.286714	0.318836	0.212957	0.334239	0.000000	0.037299	0.334769	0.259161	0.272448	0.055453	...	0.000000	0.000000	0.000000	0.000000	0.000
1678	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
1679	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
1680	0.047183	0.078299	0.000000	0.056413	0.000000	0.000000	0.051498	0.082033	0.057360	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000
1681	0.047183	0.078299	0.096875	0.075218	0.094211	0.000000	0.051498	0.000000	0.071700	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000

1682 rows × 1682 columns

Step 13: Input movie name is given here to get recommended list of Movies from the above distance matrix table

Recommendation Engine

```
In [24]: try:
             #user_inp=input('Enter the reference movie title based on which recommendations are to be made: ')
             user_inp="Godfather, The (1972)"
             inp=movies[movies['title']==user_inp].index.tolist()
             inp=inp[0]

             movies['similarity'] = ratings_matrix.iloc[inp]
             movies.columns = ['movie_id', 'title', 'release_date','similarity']
             movies.head(5)
             (movies.sort_values( ["similarity"], ascending = False )[1:10]).to_csv("Recommended_List.csv")
             p=Popen('hdfs dfs -copyFromLocal -f ./Recommended_List.csv /ml-100k/Output/',shell=True,stdout=PIPE)
             stdo,stde=p.communicate()
             print stde
         except:
             print("Sorry, the movie is not in the database!")
```

Step 14: Finally the Recommended Movies were printed based on user input

```
In [29]: print "Recommended movies based on your choice of ",user_inp ," ", movies.sort_values(["similarity"], ascending = Fa
```

```
Recommended movies based on your choice of  Godfather, The (1972)
        movie_id                                title release_date  similarity
186       187          Godfather: Part II, The (1974)  01-Jan-1974    0.665401
99        100                        Fargo (1996)  14-Feb-1997    0.646271
180       181             Return of the Jedi (1983)  14-Mar-1997    0.632097
173       174          Raiders of the Lost Ark (1981)  01-Jan-1981    0.591239
55         56                   Pulp Fiction (1994)  01-Jan-1994    0.583785
97         98        Silence of the Lambs, The (1991)  01-Jan-1991    0.578349
181       182                      GoodFellas (1990)  01-Jan-1990    0.573800
233       234                            Jaws (1975)  01-Jan-1975    0.551977
356       357  One Flew Over the Cuckoo's Nest (1975)  01-Jan-1975    0.551624
```

```
In [25]: print("enjoy your MOVIE, Thank You")
```

```
enjoy your MOVIE, Thank You
```

Thank You