**ABHILASHA’s Movie Recommender Algorithm:**

The basic underlying principle that this algorithm considers is the fact that – for a given combination of movies say (m1,m2), more are the number of common viewers who like both the movies , more similar the movies are. *Adding to this the genre of the movie* watched by the user, gives us a reasonable recommender algorithm.

**Step1:**

For a given movie combination (m1,m2) , calculate 2 things :

1. Number of common viewers of movie m1 and m2 who have liked both movies i.e. who gave a rating of > 3.0 to both movies.
2. ***Average rating of movie m2*** by the common viewers of movies m1 & m2

**Step2:**

Use the above information to create a (key,value) pair like so:

(m1) 🡪 [ (m2, no.of common viewers of m1&m2, avg. rating of m2 by common users of m1 & m2),

(m3, no.of common viewers of m1&m3, avg. rating of m3 by common users of m1 & m3),

(m5, no.of common viewers of m1&m5, avg. rating of m5 by common users of m1 & m5)

. . . . . .

]

(m2) 🡪 [ (m1, no.of common viewers of m2&m1, avg. rating of m1 by common users of m2 & m1),

(m7, no.of common viewers of m2&m7, avg. rating of m7 by common users of m2 & m7),

(m8, no.of common viewers of m2&m8, avg. rating of m8 by common users of m2 & m8)

. . . . . .

]

Example: (10, [(17, 2, 4.5), (34, 4, 4.375),(110, 1, 4.0),(11, 4, 4.125)]

**Step3:**

Order the data – there can be 2 options of doing it

**Option#1** : Order the data against each movie key , first by descending no. of common viewers, then by descending avg. rating of the second movie.

Example data from step2 becomes:

(10, [(34, 4, 4.375),(11, 4, 4.125), (17, 2, 4.5), (110, 1, 4.0)]

**Option#2 :** Order the data against each movie key, by taking in aggregate ( no.of common viewers and avg. rating of second movies )

**Pick out movies from this list which have the same genre as the movie watched by user. These are the recommended movies for a user who has watched movie1.**

**More details on RDDs at each step is added on the jupyter notebook.**

**STEP WISE MAP REDUCE ACC. TO ABOVE ALGORITHM**

**Map1 :** converts the ratings data into (**key**,**value**) ===> ( **userId** , (**movieId,ratings,timestamp**) )

**||**

**||**

**||**

**\/**

**Reduce1 :** perform ***groupByKey*** on data from Map1 to combine all values against each key into a list

**||**

**||**

**||**

**\/**

**Map2 :** Drop the key (userId) from the data from Reduce1, and create combinations of movieIds. Create (**key**,**value**) ===> ( (**movie1,movie2**) , (**ratingA1,ratingA2**))

**||**

**||**

**||**

**\/**

**Reduce2 :** perform ***groupByKey*** on data from Map2,to combine all values against a single key into a list.

**||**

**||**

**||**

**\/**

**Map3:** Convert data from Reduce2 to (**key**,**value**) ===>

((**movie1**) , (**movie2,(no.ofcommonviewersm1m2),(avg-rating of movie2 across common users**)) )

**||**

**||**

**||**

**\/**

**Reduce3 :** perform ***groupByKey*** on data from Map3,to combine all values against a single key into a list.

**||**

**||**

**||**

**\/**

[ Sort this above data using sorting algorithm of your choice. I have used : Order the RDD by number of common users first (descending) & then by descending order of avg ratings Take top 5. ]