# RASHI
# Big DATA Management

## Movie Recommendation Code

Consider the Movie Recommendation code, we wrote a brief write-up on the problem, steps needed to arrive at the solution (recommendation system), and how exactly those steps are implemented in the code.

**Brief write-up:**

In the movie recommedation system, we want to recommend movies to user on the basis of what he rated in the past. We will look into user's history and see, how much rating he gave to movies.
We will recommend him movies of same ratings(No matter good or bad)

**Steps to solve our problem:**

- Read the data from users file and map it like users, (movies, ratings)
  In the user database, we will see the movies rated by particular user and extract it.

- Then, combine the movies with their ratings for the particular user.
  Ex: if a user rated, movie1 with rating1 and movie2 with rating2. So, we will form a tupple for each user, like—— [user1,(movie1 , rating1)]

- Remove duplicate ratings if present.

- Now, join them like this (movie1,movie2),(rating1,rating2)) for same user.

- Now, Group it like, What ratings given to these two movies by N users.
  Ex: -((m1,m2),((r1,r2),(r1,r2),…,(r1,r2)))

- Calculate Statistical similarity between the above ratings.We can use correlation but In the code, we are using cosineSimilarity.

- Filter and sort top 10 movies requested by user.

**DESCRIPTION BY CODE:**

```
import sys
from pyspark import SparkConf, SparkContext
from math import sqrt


def loadMovieNames():  #function to load the movies names
    movieNames = {}    #Empty dictioary of movieNames
    with open("/home/cloudera/moviedata/itemfile.txt") as f:
        #open the itemfile.txt as f
        for line in f:  #going in each line of file f
            fields = line.split('|')
             #splitting the fields on the basis of '|'
             movieNames[int(fields[0])] = fields[1].decode('ascii',
'ignore')
#Storing  the  value  of  fields[1]  in  the  key(fields[0])  of
dictionary movieNames
    return movieNames #returning movieNames dictionary


#-------------------------------------------------------------------
-----------------------------------------


def makePairs((user, ratings)):
    (movie1, rating1) = ratings[0]
#ratings is the dictionary, storing the key as movie1 and value as
rating1
     (movie2, rating2) = ratings[1] #storing the key as movie2 and
value as rating2
    return ((movie1, movie2), (rating1, rating2))
#Now, making pair of movie1 and movie2 as key and rating1, rating2
as it's value


#-------------------------------------------------------------------
-----------------------------------------


# You might have the same two movies joined in both orders, or a
movie joined to itself, for example:
# (user, ((100,5),(200,4)))
# (user, ((200,4),(100,5)))
# Remove duplicates by filtering for the condition where movie1 <
movie2
def filterDuplicates( (userID, ratings) ):
    (movie1, rating1) = ratings[0] #movie 1 with it's rating
    (movie2, rating2) = ratings[1] #movie 2 with it's rating
    return movie1 < movie2
    #Filtering after comparison of movie1<movie2
```

```
#-------------------------------------------------------------------
-----------------------------------------


# Cosine similarity is a metric used to measure how similar the
documents are irrespective of their size.
# Mathematically, it measures the cosine of the angle between two
vectors projected in a multi-dimensional space
def computeCosineSimilarity(ratingPairs):
    numPairs = 0
    sum_xx = sum_yy = sum_xy = 0
    for ratingX, ratingY in ratingPairs:
#In the rating Pairs, we have pairs of ratings. Here we
are traversing those ratings in the pair as Rating x and rating y.
        sum_xx += ratingX * ratingX
#adding the multiplication of ratingX every time in sum_xx
        sum_yy += ratingY * ratingY
 #adding the multiplication of ratingY every time in sum_yy
         sum_xy += ratingX * ratingY    #adding the multiplication
of ratingX and ratingY every time in sum_xy
        numPairs += 1                #increment the count of Pairs

    numerator = sum_xy        #taking value of sum_xx as numerator
    denominator = sqrt(sum_xx) * sqrt(sum_yy)
#(squareroot of sum_xx)*(squareroot of sum_yy) as denominator

    score = 0   #initially score is 0
    if (denominator):
#when denominator is not 0, it will go inside if
        score = (numerator / (float(denominator)))
#Now calculating score by dividing numerator by denominatr

    return (score, numPairs)
#returning the score with the count of pairs


#-------------------------------------------------------------------
-----------------------------------------


conf                                                              =
SparkConf().setMaster("local[*]").setAppName("MovieSimilarities")
#To run a Spark application on the local/cluster, we need to set a
few configurations and parameters, this is what SparkConf helps with.
It provides configurations to run a Spark application.
#Initially, we will create a SparkConf object with SparkConf(), which
will load the values from spark.* Java system properties as well.
Now we can set different parameters using the SparkConf
object and their parameters will take priority over the
system properties.
```

```
#setMaster(value) - To set the master URL.
#setAppName(value) - To set an application name.

sc = SparkContext(conf = conf)

print "\nLoading movie names..."
nameDict = loadMovieNames()  #calling loadMovieNames function that
will return a dictionary of id's and MovieNames

data        =        sc.textFile("file:///home/cloudera/moviedata/
datafile2.txt") #Reading the textfile datafile2

ratings = data.map(lambda l: l.split()).map(lambda l: (int(l[0]),
(int(l[1]), float(l[2]))))
#  There  will  be  two  integer  and  1  float  in  a  row  of  file
datafile2.txt. For extracting this data.map(lambda l: l.split()), we
are using the map function to split the line into words, This will
return a list of words and
#then  map(lambda  l:  (int(l[0]),  (int(l[1]),  float(l[2]),  again
applying to read the words and convert them to respective formats.
This will return userID, MovieID and Ratings

joinedRatings = ratings.join(ratings)
#Join the ratings of movies done by same user

uniqueJoinedRatings    =    joinedRatings.filter(filterDuplicates)
#filter the duplicates

moviePairs  =  uniqueJoinedRatings.map(makePairs)  #making  pair
of movies and ratings done by same user.

moviePairRatings  =  moviePairs.groupByKey()  #grouping  by  pair
of movies

moviePairSimilarities                                              =
moviePairRatings.mapValues(computeCosineSimilarity).cache()
#finding the cosine similarity for the movie pairs based on the
ratings given by multiple users

if (len(sys.argv) > 1):

    scoreThreshold = 0.10 #assigning scoreThreshold = 0.10
    coOccurenceThreshold = 2 #assigning coOccurenceThreshold = 2

    movieID = int(sys.argv[1])  #typcasting sys.argv[1] in int and
then storing it in movieID

                                        filteredResults           =
moviePairSimilarities.filter(lambda((pair,sim)): \  #filtering  the
movie on the basis
                (pair[0]  ==  movieID  or  pair[1]  ==  movieID)  \
#of similarities.
```

```
                    and sim[0] > scoreThreshold and sim[1] >
coOccurenceThreshold)        ## If similarity is greater than
score threshold than only we will recommend that movie

        results = filteredResults.map(lambda((pair,sim)): (sim,
pair)).sortByKey(ascending = False).take(10)
    #sorting the similarity with movie pairs in descending order

       print "Top 10 similar movies for " +
       nameDict[movieID]
#printing top10 movies using loop
    for result in results:
        (sim, pair) = result
        similarMovieID = pair[0]
        if (similarMovieID == movieID):
            similarMovieID = pair[1]
         print nameDict[similarMovieID] + "\tscore: " + str(sim[0])
+ "\tstrength: " + str(sim[1])
```