**Q1. Consider the two data files (users.csv, transactions.csv**

from pyspark import SparkConf, SparkContext

import sys

conf = SparkConf().setMaster("local").setAppName("My App")

sc = SparkContext(conf = conf)

#Create rdd from csv file

users\_data = sc.textFile('users.csv')

transactions\_data = sc.textFile('transactions.csv')

users\_rdd = users\_data.map(lambda row : row.split(","))

transactions\_rdd = transactions\_data.map(lambda row : row.split(","))

#create key value pair, with userId as key for both users and transactions data

users = users\_rdd.map(lambda row : (int(row[0]), (row[1], row[2], row[3])))

transactions = transactions\_rdd.map(lambda row : (int(row[2]), (row[0], int(row[1]), int(row[3]), row[4])))

#flattten the joined tables

def flatten\_joined\_list(row):

user\_id = row[0]

return user\_id, row[1][0][0], row[1][0][1], row[1][0][2], row[1][1][0], row[1][1][1], row[1][1][2], row[1][1][3]

#Join by userId and flatten the list

user\_transactions = users.join(transactions)

user\_transactions\_rdd = user\_transactions.map(lambda row : flatten\_joined\_list(row))

#check how joined rdd looks like

print(user\_transactions\_rdd.take(15))

1. **Count of unique locations where each product is sold.**

#count by unique locations

count\_by\_location = user\_transactions\_rdd.map(lambda row : row[3]).countByValue()

for location, count in count\_by\_location.items() :

print(location, count)

1. **Find out products bought by each user.**

product\_by\_user = user\_transactions\_rdd.map(lambda row : (row[0],row[5]))

product\_by\_user.collect()

1. **Total spending done by each user on each product.**

product\_by\_user = user\_transactions\_rdd.map(lambda row : (row[0], row[5], row[6]))

product\_by\_user.collect()

**Q2. Consider the dataset file Olympics.csv**

#Create RDD from file, separated by tabs

olympics = sc.textFile('olympics.csv').map(lambda row : row.split("\t"))

olympics.take(5)

1. **Total medals that each country won in a particular sport (such as Gymnastics).**

medal\_count\_by\_country = olympics.map(lambda row : ((row[2], row[5]), int(row[9])))

medal\_count\_by\_country.reduceByKey(lambda key, count : key + count).sortByKey().collect()

1. **In each Olympic games, how many medals has India won?**

#Filter out rows for India

olympics\_india = olympics.filter(lambda x: 'India' in x)

#Take year and total medal count from filtered rdd and sum values using reduceByKey

count\_by\_year = olympics\_india.map(lambda x : (x[3],int(x[9])))

count\_by\_year.reduceByKey(lambda year, count : year + count).collect()

1. **Compute top 3 countries in terms of total medals by each Olympic games year.**

olympics\_by\_year = olympics.map(lambda x : ((x[3], x[2]), int(x[9])))

olympics\_by\_year.reduceByKey(lambda key, count : key + count).sortByKey().collect()

**Q3. Consider the Movie Recommendation code and problem that was discussed during the class**

In this problem, we are taking movie ratings by users and computing similarity score between two users and movies. Based on this, we need to give recommendations to users. Steps used for this –

1. Take movies and ratings from ml-100k dataset with 100,000 ratings from 1000 users on 1700 movies
2. We create pairs of movies and ratings with other movies and ratings
3. Then every movie pair is grouped and all ratings are combined in the format - ((m1,m2),((r1,r2),(r1,r2),…,(r1,r2)))
4. Calculate consine similarity of each pair and calculate coocurance score based on number of times two movies were rated with similar rating
5. Get input from user based on which particular movies, recommendations has to be given
6. And for given movie, get all similar movies and recommend

# import necessary libraries from pyspark to create sparkcontext and set hadoop cluster configurations

import sys

from pyspark import SparkConf, SparkContext

from math import sqrt

# Read pipe separated movie names and details. However only store movie id and name as key value pair in movieNames dictionary

def loadMovieNames():

movieNames = {}

with open("../movies.txt") as f:

for line in f:

fields = line.split('|')

print(fields)

movieNames[int(fields[0])] = fields[1].encode().decode()#decode('ascii', 'ignore')

return movieNames

# Re-organize movie-rating pairs such that pair of two movies is the key, and pair of ratings become the value.

This is further used to group by movie pairs

def makePairs((user, ratings)):

(movie1, rating1) = ratings[0]

(movie2, rating2) = ratings[1]

return ((movie1, movie2), (rating1, rating2))

#Remove all duplicate user, movie pair ratings. During cartesian join movie, ratings pairs gets repeated. For example -

(user1, ((movie1,rating1),(movie1,rating1))

(user1, ((movie1,rating1),(movie2,rating2))

(user1, ((movie1,rating1),(movie3,rating3))

(user1, ((movie2,rating2),(movie1,rating1))

(user1, ((movie2,rating2),(movie2,rating2))

Post applying this function, after removing duplicates, above pairs will becomes -

(user1, ((movie1,rating1),(movie2,rating2))

(user1, ((movie1,rating1),(movie3,rating3))

def filterDuplicates( (userID, ratings) ):

(movie1, rating1) = ratings[0]

(movie2, rating2) = ratings[1]

return movie1 < movie2

#Below function calculate the similarity score between two movies. Here we are using consine similarity -

- Lesser the angle between two movies, more will be the value of cos theta

- If two movies are forming positive correlation, they can be recommended to user. If they are forming negative correlation, they shouldn't be recommended at all

Refs - https://www.coursera.org/lecture/networks-illustrated/cosine-similarity-g1d3C

<https://www.machinelearningplus.com/nlp/cosine-similarity/>

def computeCosineSimilarity(ratingPairs):

numPairs = 0

sum\_xx = sum\_yy = sum\_xy = 0

for ratingX, ratingY in ratingPairs:

sum\_xx += ratingX \* ratingX

sum\_yy += ratingY \* ratingY

sum\_xy += ratingX \* ratingY

numPairs += 1

numerator = sum\_xy

denominator = sqrt(sum\_xx) \* sqrt(sum\_yy)

score = 0

if (denominator):

score = (numerator / (float(denominator)))

return (score, numPairs)

#Create single cluster spark master and worker node on localhost

conf = SparkConf().setMaster("local[\*]").setAppName("MovieSimilarities")

sc = SparkContext(conf = conf)

#Load movie names and id in RDD

print("\nLoading movie names...")

nameDict = loadMovieNames()

# Read the ratings file, and create ratings rdd - a key value pair such that key - [userId, (MovieId, Rating)]

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

ratings = data.map(lambda l: l.split()).map(lambda l: (int(l[0]), (int(l[1]), float(l[2]))))

# create user ratings pair, for each movie rated by user. makePairs function joins movies 1, 2, and 3 rated by user as-

(movie1,rating1), (movie1,rating1)

(movie1,rating1), (movie2,rating2)

(movie1,rating1), (movie3,rating3)

joinedRatings = ratings.join(ratings)

#Filter all duplicate movie pairs by applying filterDuplicates function

uniqueJoinedRatings = joinedRatings.filter(filterDuplicates)

#reorganize movie and ratings pair, to make movieId pair as key and rating pairs as value

moviePairs = uniqueJoinedRatings.map(makePairs)

#Group by movie id pairs to include all ﻿ratings as one iterable value, example –

[((2997, 3993), <pyspark.resultiterable.ResultIterable object at 0x2041f90>), ((3968, 99992), <pyspark.resultiterable.ResultIterable object at 0x2061190>), ((563, 2409), <pyspark.resultiterable.ResultIterable object at 0x20611d0>)…]

moviePairRatings = moviePairs.groupByKey()

#calculate cosine similarity values and create a map with number of times two movies were rated same along with cosine similarity value. Resultant map would look like -

﻿((160, 950), (0.98019605881960703, 3)), ((144, 8720), (1.0, 1)), ((8157, 8253), (0.99984614201001332, 2)), ((3000, 155774), (1.0, 1)), ((4447, 5989), (0.96921144065253273, 33)), ((1237, 37727), (0.88082439701565107, 2))

moviePairSimilarities = moviePairRatings.mapValues(computeCosineSimilarity).cache()

#Read the user input for movie Id

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

#set threshold for minimum match value and coccurance value for number of times rated same

scoreThreshold = 0.10

coOccurenceThreshold = 2

#store input movieId from user

movieID = int(sys.argv[1])

#Get all movie pairs which matches the movie, we want to give recommendation upon, or matches threshold for match

filteredResults = moviePairSimilarities.filter(lambda((pair,sim)): \

(pair[0] == movieID or pair[1] == movieID) \

and sim[0] > scoreThreshold and sim[1] > coOccurenceThreshold)

#Take top 10 results from above filtered map and sort it by movie Id

results = filteredResults.map(lambda((pair,sim)): (sim, pair)).sortByKey(ascending = False).take(10)

#Print all similar movie names by matching ids in movie dictionary and also show the cosine similarity index and cooccurrence value

print "Top 10 similar movies for " + nameDict[movieID]

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])