Manoj Sharma

By making use of Spark Core (i.e. without using Spark SQL) find out:

Count of unique locations where each product is sold.

>>> ##1a) Count of unique locations where each product is sold

... ##load userfile and transaction file to create rdd

... userfile = sc.textFile("/home/cloudera/users.csv")

>>> transfile = sc.textFile("/home/cloudera/transactions.csv")

>>>

>>> ##for this question we need to join the two tables as location is in

... ##first table and products in second table

... ##for joining the rdd's we need to convert them to paired RDD

... ##usermap is paired with key = userid, value = location

... usermap = userfile.map(lambda x: (x.split(",")[0], (x.split(",")[3])))

>>>

>>> ##transmap is paired with key = userid, value = product id

... transmap = transfile.map(lambda x:(x.split(",")[2], (x.split(",")[1])))

>>> ##using inner join to contain only rows where transactions happned

... joinrdd = usermap.join(transmap)

>>> ##display joined rdd

... joinrdd.collect()

[(u'19', (u'US', u'1004')), (u'19', (u'US', u'1002')), (u'1', (u'MX', u'1004')), (u'5', (u'CA', u'1005')), (u'9', (u'MX', u'1001')), (u'10', (u'CA', u'1001')), (u'17', (u'MX', u'1004')), (u'22', (u'MX', u'1001')), (u'26', (u'IN', u'1002')), (u'3', (u'FR', u'1003')), (u'7', (u'FR', u'1004')), (u'30', (u'US', u'1002')), (u'16', (u'US', u'1002')), (u'2', (u'US', u'1004')), (u'6', (u'FR', u'1003'))]

>>>

>>> ##TO COUNT the distinct locations where each product is sold

... uniloc = joinrdd.map(lambda x: x[1][1]).distinct().count()

>>> print(uniloc)

5

b) Find out products bought by each user.

>>> ##1b Find out Products bought by each user

... ## WE dont need joined rdd here, required info is present in transactions rdd

... transfile = sc.textFile("/home/cloudera/transactions.csv")

>>> ## mapping with userid and reducing with the count of products

... transmap = transfile.map(lambda x:(x.split(",")[2], 1))

>>> transreduce = transmap.reduceByKey(lambda x,y : x + y)

>>> transreduce.collect()

[(u'26', 1), (u'17', 1), (u'22', 1), (u'19', 2), (u'1', 1), (u'3', 1), (u'5', 1), (u'7', 1), (u'9', 1), (u'2', 1), (u'10', 1), (u'6', 1), (u'30', 1), (u'16', 1)]

>>>

c) Total spending done by each user on each product.

>>> ##1c Total spending done by each user on each product

... ## Again no use of joined RDD, we can get this from transactions rdd

... ## mapping with userid and reducing with price of product

... ## we need to convert price to int to enable aggregation

... transfile = sc.textFile("/home/cloudera/transactions.csv")

>>> transmap = transfile.map(lambda x:(x.split(",")[2], int((x.split(",")[3]))))

>>> transreduce = transmap.reduceByKey(lambda x,y : x + y)

>>> transreduce.collect()

[(u'26', 149), (u'17', 129), (u'22', 99), (u'19', 278), (u'1', 129), (u'3', 89), (u'5', 199), (u'7', 129), (u'9', 99), (u'2', 129), (u'10', 99), (u'6', 89), (u'30', 149), (u'16', 149)]

>>>

scala> //loading olympics data into rdd

scala> val olydata = sc.textFile("/home/cloudera/olympics.csv")

olydata: org.apache.spark.rdd.RDD[String] = /home/cloudera/olympics.csv MapPartitionsRDD[1] at textFile at <console>:24

scala> //defining a map for medals with Key = sports,country), value = medals

scala> //splitting the input rdd with tab the delimiter

scala> val Medalsrdd = olydata.map(x => {

| val datalist = x.split("\t")

| val country = datalist(2)

| val sports = datalist(5)

| val medals = datalist(9).toInt

| ((sports,country),medals)

| }).reduceByKey((x,y) => x +y).sortByKey()

Medalsrdd: org.apache.spark.rdd.RDD[((String, String), Int)] = ShuffledRDD[6] at sortByKey at <console>:31

scala> //reduce by medals counts, to get total for medals for each country

scala> //sort by sports to give categorical view for each sports

scala> //printing the top 10 results

scala> Medalsrdd.take(10).foreach(println)

((Alpine Skiing,Austria),27)

((Alpine Skiing,Croatia),9)

((Alpine Skiing,Czech Republic),1)

((Alpine Skiing,Finland),1)

((Alpine Skiing,France),6)

((Alpine Skiing,Germany),4)

((Alpine Skiing,Italy),4)

((Alpine Skiing,Norway),9)

((Alpine Skiing,Slovenia),2)

((Alpine Skiing,Sweden),8)

scala> //loading olympics data into rdd

scala> val olydata=sc.textFile("/home/cloudera/olympics.csv")

olydata: org.apache.spark.rdd.RDD[String] = /home/cloudera/olympics.csv MapPartitionsRDD[8] at textFile at <console>:24

scala> //printing olympics data count and first record

scala> olydata.count

res1: Long = 8618

scala> olydata.first

res2: String = Michael Phelps 23 United States 2008 08-24-08 Swimming 8 0 0 8

scala> //defining a filter for indis's medals

scala> //splitting the input rdd with tab the delimiter

scala> val indiardd = olydata.filter(x => {

| val country = x.split("\t")(2)

| country == "India"

| })

indiardd: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[9] at filter at <console>:25

scala> //printing India's olympics data count and first record

scala> indiardd.first

res3: String = Yogeshwar Dutt 29 India 2012 08-12-12 Wrestling 0 0 1 1

scala> indiardd.count

res4: Long = 11

scala> indiardd.take(20).foreach(println)

Yogeshwar Dutt 29 India 2012 08-12-12 Wrestling 0 01 1

Sushil Kumar 29 India 2012 08-12-12 Wrestling 0 10 1

Sushil Kumar 25 India 2008 08-24-08 Wrestling 0 01 1

Karnam Malleswari 25 India 2000 10-01-00 Weightlifting 00 1 1

Vijay Kumar 26 India 2012 08-12-12 Shooting 0 10 1

Gagan Narang 29 India 2012 08-12-12 Shooting 0 01 1

Abhinav Bindra 25 India 2008 08-24-08 Shooting 1 00 1

Rajyavardhan Rathore 34 India 2004 08-29-04 Shooting 01 0 1

M. C. Mary Kom 29 India 2012 08-12-12 Boxing 0 0 11

Vijender Singh 22 India 2008 08-24-08 Boxing 0 0 11

Saina Nehwal 22 India 2012 08-12-12 Badminton 0 01 1

scala> //defining a map for medals with Key = years,value = medalcounts

scala> val yearmap = indiardd.map(x => {

| (x.split("\t")(3), 1)

| })

yearmap: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[10] at map at <console>:25

scala> //reduce the medal counts and sort by year

scala> val yearmedals = yearmap.reduceByKey((x,y) => x + y).

| map(year => (-year.\_2, year.\_1)).

| sortByKey().

| map(year => (year.\_2, -year.\_1))

yearmedals: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[16] at map at <console>:28

scala>

scala> yearmedals.count()

res6: Long = 4

scala> yearmedals.take(10).foreach(println)

(2012,6)

(2008,3)

(2004,1)

(2000,1)

scala> //loading olympics data into rdd

scala> val olydata = sc.textFile("/home/cloudera/olympics.csv")

olydata: org.apache.spark.rdd.RDD[String] = /home/cloudera/olympics.csv MapPartitionsRDD[18] at textFile at <console>:24

scala> //defining a map for medals with Key = sports,country), value = medals

scala> //splitting the input rdd with tab the delimiter

scala> //defining a map for medals with Key = years,country,value = medalcounts

scala> val Medalsrdd = olydata.map(x => {

| val datalist = x.split("\t")

| val country = datalist(2)

| val year = datalist(3)

| val medals = datalist(9).toInt

| ((year,country),medals)

| }).reduceByKey((x,y) => x +y).map(a => (-a.\_2,a.\_1)).sortByKey().map(a => (a.\_2, -a.\_1))

Medalsrdd: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[25] at map at <console>:31

scala>

scala> Medalsrdd.take(10).foreach(println)

((2008,United States),317)

((2004,United States),265)

((2012,United States),254)

((2000,United States),243)

((2004,Russia),191)

((2000,Russia),187)

((2008,China),184)

((2000,Australia),183)

((2004,Australia),156)

((2004,Germany),149)

scala>

#Import Sys library

#Import SparkConf and SparkContext from pyspark to setup sc

#Import sqrt from library math

import sys

from pyspark import SparkConf, SparkContext

from math import sqrt

#Function defined to load movie names from u.item file

#sample line from u.item

#1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|

#fields is splitting the string by '|' and taking second element as movie name

#index for movie names is set as starting from 0

def loadMovieNames():

movieNames = {}

with open("/home/cloudera/moviedata/itemfile.txt") as f:

for line in f:

fields = line.split('|')

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

return movieNames

#This function is used to make pairs for users and ratings

def makePairs((user, ratings)):

(movie1, rating1) = ratings[0]

(movie2, rating2) = ratings[1]

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

def filterDuplicates( (userID, ratings) ):

(movie1, rating1) = ratings[0]

(movie2, rating2) = ratings[1]

return movie1 < movie2

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)

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

sc = SparkContext(conf = conf)

print "\nLoading movie names..."

nameDict = loadMovieNames()

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]))))

joinedRatings = ratings.join(ratings)

uniqueJoinedRatings = joinedRatings.filter(filterDuplicates)

moviePairs = uniqueJoinedRatings.map(makePairs)

moviePairRatings = moviePairs.groupByKey()

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

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

scoreThreshold = 0.10

coOccurenceThreshold = 2

movieID = int(sys.argv[1])

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

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

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

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

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])