Lab on pyspark : Saad Lahlali

October 13, 2021

1 Settings

[ ]:
```
import pyspark
sc = pyspark.SparkContext('local[4]',appName="Spark Lab Session")
```

2 First steps with Spark

2.1 First RDD

[ ]:
```
rdd = sc.parallelize(range(3000))
```

[ ]:
```
rdd.take(10)
```

[ ]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.2 Computing the sum of cubes

[ ]:
```
rdd2 = rdd.map(lambda x : x**3)
```

[ ]:
```
rdd2.take(10)
```

[ ]:
```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

[ ]:
```
rdd2.sum()
```

[ ]:
```
20236502250000
```

2.3 Last digits of elements in C

[ ]:
```
last_i = rdd2.map(lambda x : (x%10, 1)).reduceByKey(lambda a, b: a + b)
```

[ ]:
```
last_i.collect()
```

[ ]:
```
[(0, 300),
 (8, 300),
 (4, 300),
 (1, 300),
 (5, 300),
```

```
 (9, 300),
 (6, 300),
 (2, 300),
 (7, 300),
 (3, 300)]
```

2.4 Digits of C

[ ]:
```
digits_C = rdd2.flatMap(lambda i : [e for e in str(i)]).map(lambda x : (x,1)).
↪reduceByKey(lambda a, b: a + b)
```

[ ]:
```
digits_C.collect()
```

[ ]:
```
[('4', 2762),
 ('7', 2787),
 ('6', 2713),
 ('3', 2814),
 ('0', 3127),
 ('1', 3667),
 ('8', 2639),
 ('9', 2521),
 ('2', 3294),
 ('5', 2653)]
```

3 Using the Movie Lens dataset

3.1 Getting the dataset

[ ]:
```
import re

future_pattern = re.compile("""([^,"]+|"[^"]+")(?=,|$)""")

def parseCSV(line):
  return future_pattern.findall(line)

ratingsFile = sc.textFile("/content/drive/MyDrive/ratings.csv").map(parseCSV)
moviesFile = sc.textFile("/content/drive/MyDrive/movies.csv").map(parseCSV)
```

[ ]:
```
ratingsFile.take(2)
```

[ ]:
```
[['userId', 'movieId', 'rating', 'timestamp'], ['1', '1', '4.0', '964982703']]
```

[ ]:
```
moviesFile.take(2)
```

[ ]:
```
[['movieId', 'title', 'genres'],
 ['1', 'Toy Story (1995)', 'Adventure|Animation|Children|Comedy|Fantasy']]
```

3.2 Cleaning data

[ ]:
```
def del_header(l):
  # deleting the line if header of ratings or movies
  if l[0]=='userId' or l[0]=='movieId':
    return
  return l
```

[ ]:
```
ratings = ratingsFile.filter(del_header).map(lambda l : [l[0], l[1],␣
↪float(l[2]), l[3]]) # ratings : str => float
movies = moviesFile.filter(del_header)
```

[ ]:
```
ratings.take(4)
```

[ ]:
```
[['1', '1', 4.0, '964982703'],
 ['1', '3', 4.0, '964981247'],
 ['1', '6', 4.0, '964982224'],
 ['1', '47', 5.0, '964983815']]
```

[ ]:
```
movies.take(4)
```

[ ]:
```
[['1', 'Toy Story (1995)', 'Adventure|Animation|Children|Comedy|Fantasy'],
 ['2', 'Jumanji (1995)', 'Adventure|Children|Fantasy'],
 ['3', 'Grumpier Old Men (1995)', 'Comedy|Romance'],
 ['4', 'Waiting to Exhale (1995)', 'Comedy|Drama|Romance']]
```

3.3 10 best movies of all times

[ ]:
```
best = sc.parallelize(ratings.map(lambda l : (l[1], (l[2], 1))).
↪reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])).mapValues(lambda v: v[0]/
↪v[1]).sortBy(lambda x: x[1], ascending=False).take(10))
```

[ ]:
```
best.collect() # (key=movieID, value=rating)
```

[ ]:
```
[('6835', 5.0),
 ('1151', 5.0),
 ('1631', 5.0),
 ('102217', 5.0),
 ('27523', 5.0),
 ('53', 5.0),
 ('1140', 5.0),
 ('8238', 5.0),
 ('47736', 5.0),
 ('53355', 5.0)]
```

3.4 Ordered list of movies with names

[ ]:
```
mov = movies.map(lambda l : (l[0], l[1]))
ordered = mov.join(best).map(lambda l: (l[1][0],l[1][1]))
```

[ ]:
```
ordered.collect()
```

[ ]:
```
[('Lamerica (1994)', 5.0),
 ('My Sassy Girl (Yeopgijeogin geunyeo) (2001)', 5.0),
 ('Entertaining Angels: The Dorothy Day Story (1996)', 5.0),
 ('Lesson Faust (1994)', 5.0),
 ('"Chump at Oxford, A (1940)"', 5.0),
 ('Sun Alley (Sonnenallee) (1999)', 5.0),
 ('Bill Hicks: Revelations (1993)', 5.0),
 ('"Assignment, The (1997)"', 5.0),
 ('Alien Contamination (1980)', 5.0),
 ('Little Murders (1971)', 5.0)]
```

3.5 Better ordered list

3.5.1 Metric 1

[ ]:
```
best_metric_1 = sc.parallelize(ratings.map(lambda l : (l[1], (l[2], 1))).
↪reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])).mapValues(lambda v: v[0]/
↪(v[1]+1)).sortBy(lambda x: x[1], ascending=False).take(10))
```

[ ]:
```
best_metric_1.collect()
```

[ ]:
```
[('318', 4.415094339622642),
 ('858', 4.266839378238342),
 ('1104', 4.261904761904762),
 ('2959', 4.2534246575342465),
 ('1221', 4.226923076923077),
 ('750', 4.224489795918367),
 ('177593', 4.222222222222222),
 ('50', 4.217073170731707),
 ('1213', 4.216535433070866),
 ('260', 4.214285714285714)]
```

[ ]:
```
mov.join(best_metric_1).map(lambda l: (l[1][0],l[1][1])).collect()
```

[ ]:
```
[('"Godfather: Part II, The (1974)"', 4.226923076923077),
 ('"Godfather, The (1972)"', 4.266839378238342),
 ('Goodfellas (1990)', 4.216535433070866),
 ('Star Wars: Episode IV - A New Hope (1977)', 4.214285714285714),
 ('"Shawshank Redemption, The (1994)"', 4.415094339622642),
 ('Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)',
  4.224489795918367),
 ('"Usual Suspects, The (1995)"', 4.217073170731707),
 ('"Streetcar Named Desire, A (1951)"', 4.261904761904762),
```

```
('Fight Club (1999)', 4.2534246575342465),
('"Three Billboards Outside Ebbing, Missouri (2017)"', 4.222222222222222)]
```

3.5.2 Metric 2

[ ]:
```
from math import log
```

[ ]:
```
best_metric_2 = sc.parallelize(ratings.map(lambda l : (l[1], (l[2], 1))).
↪reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])).mapValues(lambda v: (v[0]/
↪v[1])*log(v[1]+1)).sortBy(lambda x: x[1], ascending=False).take(10))
```

[ ]:
```
best_metric_2.collect()
```

[ ]:
```
[('318', 25.5202528120611),
 ('356', 24.148197375717082),
 ('296', 24.049620750363033),
 ('2571', 23.608551552923746),
 ('593', 23.44799544544623),
 ('260', 23.395433031622037),
 ('2959', 23.0271574143306),
 ('527', 22.807237413912503),
 ('1196', 22.58143834749173),
 ('858', 22.57200713834986)]
```

[ ]:
```
mov.join(best_metric_2).map(lambda l: (l[1][0],l[1][1])).collect()
```

[ ]:
```
[("Schindler's List (1993)", 22.807237413912503),
 ('"Silence of the Lambs, The (1991)"', 23.44799544544623),
 ('"Godfather, The (1972)"', 22.57200713834986),
 ('"Matrix, The (1999)"', 23.608551552923746),
 ('Pulp Fiction (1994)', 24.049620750363033),
 ('Star Wars: Episode V - The Empire Strikes Back (1980)', 22.58143834749173),
 ('Star Wars: Episode IV - A New Hope (1977)', 23.395433031622037),
 ('"Shawshank Redemption, The (1994)"', 25.5202528120611),
 ('Forrest Gump (1994)', 24.148197375717082),
 ('Fight Club (1999)', 23.0271574143306)]
```

4 Re-implementing K-Means

4.1 Loading data

[ ]:
```
pts = []
ground_truth = dict()
with open("/content/drive/MyDrive/SZRd1P4Lb5S_data.tsv","r") as f:
  for l in f.readlines():
    data=l.split("\t")
    pts.append(tuple(map(float,data[:-1])))
    ground_truth[pts[-1]] = int(data[-1].rstrip())
```

```
    ptsRDD = sc.parallelize(pts)
```

[ ]:
```
ptsRDD.take(1)
```

[ ]:
```
[(-0.1809699459806945,
  2.192028953740528,
  3.504043825943004,
  -1.0896302460725749,
  1.6057579035478744,
  2.1445200774212987,
  1.1218850520596748,
  1.4346564576609544,
  2.084084922865424,
  -0.13070819907001963)]
```

The points are in a space of 10 dimensions.

4.2 Initialization

[ ]:
```
# Setting the number of clusters
K = 7
```

[ ]:
```
from random import random
```

[ ]:
```
ini = ptsRDD.map(lambda coord : (coord, random())).sortBy(lambda x: x[1],␣
 ↪ascending=False).map(lambda l: l[0])
```

[ ]:
```
centroids = ini.take(K)
```

[ ]:
```
print(ini_centroids[0])
```

```
(2.2539289282176744, 2.0138731406718007, 2.9060400593588716, 2.750413972149768,
2.85117895363233, 0.6046272472483172, 2.0602073029812087, 4.099791334412553,
1.879706175730693, 2.5505453006455454)
```

4.3 Step method

[ ]:
```
def find_closest(coord):

  #finding the closest centroid to the current point
  index_closest, min_dist = -1, 99999999999999
  for i in range(len(centroids)):
    s = 0
    for j in range(len(coord)):
      s += (coord[j]-centroids[i][j])**2
    if s < min_dist:
      index_closest, min_dist = i, s

  return (index_closest, (coord, 1))
```

We first map by setting as key the index of the closest centroid to each point and as value the tuple containing the coordinates of the point and the integer 1.

[ ]:
```
centroids_step = ptsRDD.map(find_closest)
centroids_step.take(2)
```

[ ]:
```
[(3,
  ((-0.1809699459806945,
    2.192028953740528,
    3.504043825943004,
    -1.0896302460725749,
    1.6057579035478744,
    2.1445200774212987,
    1.1218850520596748,
    1.4346564576609544,
    2.084084922865424,
    -0.13070819907001963),
   1)),
 (2,
  ((1.6258456835886963,
    1.664416507223232,
    1.8254395674483501,
    2.5452178986004688,
    1.139116728970604,
    1.0591336453068587,
    1.5590047150797508,
    -0.24174010719233685,
    2.0106206364597496,
    0.5886273087707666),
   1))]
```

Then we reduce by key by summing up the coordinates in each dimension of each point associated to each centroid, the second element of the value contains the number of closest points to the centroid.

[ ]:
```
centroids_step = centroids_step.reduceByKey(lambda a,b: ([x+y for x,y in␣
 ↪zip(a[0],b[0])], a[1]+b[1]))
centroids_step.take(1)
```

[ ]:
```
[(4,
  ([1845.2936545264351,
    1192.8779783035527,
    2287.1208007293785,
    2017.9660491256982,
    1800.546120345019,
    393.5026523007212,
    1569.9642699171723,
    1417.139512489974,
    507.2340321437089,
    1716.1848464419786],
```

```
714))]
```

After that, we calculate the average according to the first formula given in the lab.

[ ]:
```
centroids_step = centroids_step.mapValues(lambda v: [(x/v[1])*log(v[1]+1) for x␣
↪in v[0]])
centroids_step.take(1)
```

[ ]:
```
[(4,
  [16.985702061327896,
   10.980295675586218,
   21.05266682306167,
   18.575130303106477,
   16.5738064902787,
   3.6221437145935855,
   14.451328800882315,
   13.044595628150914,
   4.6690271351798165,
   15.797271297348194])]
```

Finaly, we output the new coordinates of the centroids.

[ ]:
```
centroids_step = centroids_step.map(lambda l: l[1])
centroids_step.take(1)
```

[ ]:
```
[[16.985702061327896,
  10.980295675586218,
  21.05266682306167,
  18.575130303106477,
  16.5738064902787,
  3.6221437145935855,
  14.451328800882315,
  13.044595628150914,
  4.6690271351798165,
  15.797271297348194]]
```

4.4 Getting results

[ ]:
```
def KMeansRDD(R,K,I):
  centroids = R.map(lambda coord : (coord, random())).sortBy(lambda x: x[1],␣
↪ascending=False).map(lambda l: l[0]).take(K)
  for i in range(I):
    centroids = R.map(find_closest).reduceByKey(lambda a,b: ([x+y for x,y in␣
↪zip(a[0],b[0])], a[1]+b[1])).mapValues(lambda v: [(x/v[1])*log(v[1]+1) for x␣
↪in v[0]]).map(lambda l: l[1]).take(K)
  return centroids
```

[ ]:
```
final_centroids = KMeansRDD(ptsRDD, 7, 100)
```

[ ]:
```
final_centroids[0]
```

[ ]:
```
[16.985702061327896,
 10.980295675586218,
 21.05266682306167,
 18.575130303106477,
 16.5738064902787,
 3.6221437145935855,
 14.451328800882315,
 13.044595628150914,
 4.6690271351798165,
 15.797271297348194]
```

4.5 Measuring the performance of your algorithm

[ ]:
```
import numpy as np
```

[ ]:
```
def predict(coord):

  #finding the closest centroid to the current point
  index_closest, min_dist = -1, 99999999999999
  for i in range(len(centroids)):
    s = 0
    for j in range(len(coord)):
      s += (coord[j]-centroids[i][j])**2
    if s < min_dist:
      index_closest, min_dist = i, s

  return (coord, index_closest)
```

[ ]:
```
def proj(i):
  l = [(2,6), (6,5), (0,4), (1,3), (3, 2), (4,1),(5,0)]
  for v in l:
    if i==v[0]:
      return v[1]
```

[ ]:
```
pred = ptsRDD.map(predict)

M = np.zeros((7, 7), dtype=int)

for prediction in pred.collect():
  M[proj(prediction[1])][ground_truth[prediction[0]]]+=1

print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in M]))
```

```
291     51      462     12      11      78      17
1       201     0       0       509     0       3
396     365     535     0       0       0       3
505     13      7       1106    4       24      0
76      236     0       0       870     24      0
```

```
13      184     0       0       0       1279    0
102     390     427     335     31      10      1429
```

We get good results in general since the highest values are mostly in the diagonal and we can notice multiple 0 outside. However the is some confusion between some classes since for the class 2 gets often mistaken with the classes 0 or 6.

[ ]:
```
!jupyter nbconvert --to PDF "Lab on pyspark : Saad Lahlali.ipynb"
```

```
[NbConvertApp] Converting notebook Lab on pyspark : Saad Lahlali.ipynb to PDF
```