Universidade Federal do ABC
Inteligência na Web e Big Data
2019.Q3

Lista de Exercícios – C2

Exercício 1

Dada a matriz binária em que cada linha é um documento e cada coluna um token:

	A	B	C	D	E	F	G
D1	0	0	1	1	0	1	0
D2	1	0	1	0	0	1	0
D3	0	1	0	1	1	0	1
D4	0	0	0	1	1	1	0
D5	0	1	0	1	1	0	0

E as seguintes funções de hash:

$$h1(x) = (3 * x + 1) \mod 13$$
$$h2(x) = (4 * x + 3) \mod 13$$
$$h3(x) = (6 * x + 2) \mod 13$$
$$h4(x) = (10 * x + 4) \mod 13$$

Calcule a assinatura LSH para 2 faixas e 2 linhas.

Primeiro calculamos h1,h2,h3,h4 para cada índice de atributos (de 0 a 6):

```
h1 = [1,4,7,10,0,3,6]
h2 = [3,7,11,2,6,10,1]
h3 = [2,8,1,7,0,6,12]
h4 = [4,1,11,8,5,2,12]
```

Em seguida, para cada documento, filtramos os valores de h que correspondem a um elemento com valor 1 no vetor binário e escolhemos o menor.

Exemplo, para D1 e h1:

```
[1,4,7,10,0,3,6]
[0,0,1,1,0,1,0]
```

temos [7,10,3] e o menor valor é 3.

Feito isso, temos o vetor de minHash para cada documento:

```
D1 = [3,2,1,2]
D2 = [1,3,1,2]
D3 = [0,1,0,1]
D4 = [0,2,0,2]
D5 = [0,2,0,1]
```

Separando em duas faixas e bandas temos:

```
D1 = [(3,2), (1,2)]
D2 = [(1,3), (1,2)]
D3 = [(0,1), (0,1)]
D4 = [(0,2), (0,2)]
D5 = [(0,2), (0,1)]
```

Na primeira faixa temos colisão entre D4 e D5, na segunda faixa entre D1 e D2, e D3 e D5.

Exercício 2

Desenvolva um algoritmo MapReduce/Spark para o algoritmo Edit-KNN. O algoritmo consiste em encontrar objetos candidatos a serem desconsiderados para o KNN pois seus K-Vizinhos mais próximos possuem a mesma classe que ele. A entrada do algoritmo consiste nos dados no formato de tupla $(Int, [Double])$, sendo o primeiro item o índice do objeto, o segundo seus valores e o último a classe. O resultado final deve ser no formato $(Int, Bool)$, sendo o primeiro o índice do objeto e o segundo o valor *True* se o objeto pode ser desconsiderado, e *False* caso contrário.

Exemplo de Entrada (K=3):

```
(1, [0, 0], '+'), (2, [0, 1], '+'), (3, [1, 0], '+'), (4, [1, 1], '+'),
(5, [0.5, 0.5], '+'), (6, [9, 9], '-'), (7, [8, 8], '-'), (8, [9, 8], '-'),
(9, [8, 9], '-'), (10, [8.5, 8.5], '-'),
(11, [1.1, 1.1], '-'), (12, [9.2, 9.2], '+')
```

Exemplo de Saída:

```
(1, True), (2, True), (3, True), (4, False),
(5, True), (6, False), (7, True), (8, True),
(9, True), (10, True), (11, False), (12, False)
```

```
import pyspark
import numpy as np
import logging
from scipy.spatial.distance import euclidean

sc= pyspark.SparkContext.getOrCreate('local[*]')

objetos = [(1, [0, 0], '+'), (2, [0, 1], '+'),
           (3, [1, 0], '+'), (4, [1, 1], '+'),
           (5, [0.5, 0.5], '+'),
           (11, [1.1, 1.1], '-'),
           (6, [9, 9], '-'), (7, [8, 8], '-'),
           (8, [9, 8], '-'), (9, [8, 9], '-'),
           (10, [8.5, 8.5], '-'),
           (12, [9.2, 9.2], '+')]

def pares(v):
    key, *values =  v
    i = key
    for j in range(1,N+1):
        if j == key: continue
        k = min(i,j) * N + max(i,j)
        yield (k, [i, values])

def computa_dist(vals):
    x,y = vals[1]
    i, xi = x
    j, xj = y
    x0, x0c = xi[0], xi[1]
    x1, x1c = xj[0], xj[1]
    dst = euclidean(x0, x1)
    agree = x0c == x1c
    yield (i, (dst, agree))
    yield (j, (dst, agree))

def vizinhos_concordam(dists):
    k, vals = dists
    concordam = [x[1] for x  in vals]
    logging.error(str(k) + " --- " + str(concordam))
    return k, np.all(concordam)

def mantem_menores_seq(x, y):
    if len(x) < K:
```

```
        x.append(y)
    else:
        imax = 0
        for i in range(K):
            if x[i][0] > x[imax][0]:
                imax = i
        if y[0] < x[imax][0]:
            x[imax] = y[:]
    return x


def mantem_menores_comb(x, y):
    x.extend(y)
    ret = sorted(x)[:K]
    return ret



N=len(objetos)
K=3
sc.broadcast(N)
sc.broadcast(K)
rdd = sc.parallelize(objetos)
prs = rdd.flatMap(pares)
dst = prs.groupByKey().flatMap(computa_dist)
kviz = dst.aggregateByKey([], mantem_menores_seq,
                                mantem_menores_comb)
passiveis_remocao = kviz.map(vizinhos_concordam)

print(passiveis_remocao.collect())
```