CS-562-Homework3

April 23, 2020

1 Problem 1. (FastMap) [20 Points]

Using the **FastMap** algorithm to embed (map) the following 5-dimensional points into 3-dimensional points. Use the Euclidian distance (L2 norm) as the distance between the points in the original 5-dimensional space. Show the 3-d points of the mapping for each point.

p1: (5.1, 3.5, 1.4, 0.2, 3.0)
p2: (4.9, 3.0, 1.4, 0.2, 4.0)
p3: (4.7, 3.2, 1.3, 0.2, 2.0)
p4: (6.9, 3.1, 4.9, 1.5, 2.0)
p5: (5.5, 2.3, 4.0, 1.3, 3.0)
p6: (6.5, 2.8, 4.6, 1.5, 5.0)
p7: (7.2, 3.0, 5.8, 1.6, 7.0)
p8: (7.4, 2.8, 6.1, 1.9, 8.0)

[1]:
```
import numpy as np
import pandas as pd
from scipy.spatial import distance
import math
import matplotlib.pyplot as plt

# global variables
points = []
target_dimension = 3
dist_array = np.zeros((8, 8), dtype = np.float)
mapped = np.zeros((8, target_dimension), dtype = np.float)
pivots = np.zeros((2, 4), dtype = np.int)
colNum = 0


def preprocess():
    '''
    Read data and append to a global matrix
    '''
    O1 = [5.1, 3.5, 1.4, 0.2, 3.0]
    O2 = [4.9, 3.0, 1.4, 0.2, 4.0]
    O3 = [4.7, 3.2, 1.3, 0.2, 2.0]
    O4 = [6.9, 3.1, 4.9, 1.5, 2.0]
```

```
    O5 = [5.5, 2.3, 4.0, 1.3, 3.0]
    O6 = [6.5, 2.8, 4.6, 1.5, 5.0]
    O7 = [7.2, 3.0, 5.8, 1.6, 7.0]
    O8 = [7.4, 2.8, 6.1, 1.9, 8.0]
    points.append(O1)
    points.append(O2)
    points.append(O3)
    points.append(O4)
    points.append(O5)
    points.append(O6)
    points.append(O7)
    points.append(O8)

    return None


def dist(obj_a, obj_b):
    '''
        return L2 distance
    '''
    dis = 0
    for i in range(len(obj_a)):
        dis += (obj_a[i] - obj_b[i])**2
    return math.sqrt(dis)


def init_distances():
    '''
        create distance matrix: dist_array[N][N]
        dist_array[i][j] is the distance between object i and j in original space
    '''
    for x in range(0, 8):
        for y in range(0, 8):
            dist_array[x][y] = dist(points[x], points[y])

    return None


def choose_dist_obj(distances):
    '''
        find two points whose distance is the longest
    '''
    # randomly choose object b
    obj_b = np.random.randint(0,7)

    # Iterate until convergence
    while True:
```

```
        # use the distance matrix

        # find the farthest object from object b
        farthest = max(distances[obj_b])
        obj_a = distances[obj_b].tolist().index(farthest)

        # find the farthest object from object a
        tmp = max(distances[obj_a])
        tmpObj = distances[obj_a].tolist().index(tmp)

        # when the farthest object of object a is exatly object b,
        # it converges
        if (tmpObj == obj_b):
            break
        else:
            obj_b = tmpObj

    # return a smaller object id
    if obj_a < obj_b:
        return (obj_a, obj_b)
    else:
        return (obj_b, obj_a)


def fastMap(k, distances):
    '''
        k: target dimension
        distances: distance matrix or last projection
        This function will be called recursively.
    '''
    global colNum

    # recursion exit point
    if k <= 0:
        return None
    else:
        pass
        # colNum += 1

    # choose pivot object
    pivots = choose_dist_obj(distances)

    a = pivots[0]   # pivot a
    b = pivots[1]   # pivot b

    farthest = distances[a][b]
    if farthest == 0:
```

```
        for i in range(len(points)):
            mapped[i][colNum] = 0

    # project objects on line(Oa, Ob)
    for i in range(0, len(points)):
        temp = 0
        if i == a:
            mapped[i][colNum] = 0
        elif i == b:
            mapped[i][colNum] = farthest
        else:
            # cosine law
            temp = ((distances[a][i]**2) + (farthest**2) - (distances[b][i]**2))/
↪(2 * farthest)
            mapped[i][colNum] = temp

    # update distance matrix
    # projection will be the new distance matrix after dimension reduction
    projection = np.zeros((8, 8))
    for i in range (8):
        for j in range (8):
            # dimensional reduction
            tmp = (distances[i][j] ** 2) - ((mapped[i][colNum] -␣
↪mapped[j][colNum]) ** 2)
            projection[i][j] = np.sqrt(np.absolute(tmp))

    colNum += 1

    # recursion
    fastMap(k-1, projection)

    return None


# stress function
def get_stress(original_space_data, target_space_data):
    '''
        if the dimensional reduction maintains the dissimilarity between objects,
        this return value would be low.
    '''
    denominator = 0
    numerator = 0
    for i in range(0, len(original_space_data)-1):
        for j in range(i+1, len(original_space_data)):
            temp = dist(original_space_data[i], original_space_data[j])
            denominator += temp**2
```

```
            numerator += (temp - dist(target_space_data[i],␣
 ↪target_space_data[j]))**2
    return numerator/denominator


def main():
    preprocess()
    data = pd.DataFrame(np.array(points))
    init_distances()
    fastMap(target_dimension, dist_array)
    print('Answer:')
    print(mapped)
    print()
    print('stress function: ', get_stress(points, mapped))
    return None


main()
```

```
Answer:
[[0.89321455 0.77847574 0.        ]
 [1.57272992 0.         0.44752492]
 [0.         1.26154386 0.47790845]
 [3.05781916 4.44347949 0.44752492]
 [2.80330103 2.66288887 1.53966125]
 [4.93068837 2.04414852 0.68393317]
 [7.30058826 1.64592167 0.35510119]
 [8.32946577 1.26154386 0.47790845]]

stress function:  1.7385764793240135e-06
```

2 Problem 3 (MapReduce and databases) [30 Points]

1. In the form of relational algebra implemented in SQL, relations are not sets, but bags; that is, tuples are allowed to appear more than once. There are extended definitions of union, intersection, and difference for bags, which we shall define below. Write MapReduce algorithms for computing the following operations on bags R and S:
    1. Bag Union, defined to be the bag of tuples in which tuple t appears the sum of the numbers of times it appears in R and S.
    2. Bag Intersection, defined to be the bag of tuples in which tuple t appears the minimum of the numbers of times it appears in R and S.
    3. Bag Difference, defined to be the bag of tuples in which the number of times a tuple t appears is equal to the number of times it appears in R minus the number of times it appears in S. A tuple that appears more times in S than in R does not appear in the difference.

2.0.1 Answer:

2.0.2 Union

```
Mapper(key null, Bag bag):
    // bag is R or S.
    for each tuple t in bag:
        emit(t, bag)

Reducer(tutple t, Iterator bags):
    for each bag in bags:
        emit(t,t)
```

2.0.3 Intersection

```
Mapper(key null, Bag bag):
    for each tuple t in bag:
        emit(t, bag)         // it would be (ti, R) or (ti, S)

Reducer(tuple t, Iterator bags):
    for each bag in bags:
        emit((t, bag), 1)    // it would be ((ti, R), 1) or ((ti, S), 1)

Mapper((tuple t, Bag bag), Iterator values):
    // it would receive (ti, R), [1, 1, ... , 1]; (ti, S), [1, 1, ... , 1]
    emit(t, (bag, sum(values)))

Reducer(tuple t, Iterator (Bag bag, int count)):
    // count is the number of t's appearance times in different bags
    for i in range(min(count)):
        emit(t,t)
```

2.0.4 Difference

```
Mapper(key null, Bag bag):
    for each tuple t in bag:
        emit(t, bag)

Reducer(tuple t, Iterator bags):
    for each bag in bags:
        emit((t, bag), 1)    // it would be ((ti, R), 1) or ((ti, S), 1)

Mapper((tuple t, Bag bag), Iterator values):
    // it would receive (ti, R), [1, 1, ... , 1]; (ti, S), [1, 1, ... , 1]
    emit(t, (bag, sum(values)))
```

```
Reducer(tuple t, Iterator (Bag bag, int count)):
    // count is the number of t's appearance times in different bags
    if the list is [R,count]:
        for i in range(count):
            emit(t,t)
    else if the list is [(R,count_R), (S,count_S)]:
        if count_R > count_S:
            for i in range(count_R-count_S):
                emit(t,t)
    else if the list is [(S,count_S), (R,count_R)]:
        if count_R > count_S:
            for i in range(count_R-count_S):
                emit(t,t)
    else:
        emit nothing
```

2. The relational-algebra operation

$$R(A, B) \bowtie_{B<C} S(C, D)$$

produces all tuples (a, b, c, d) such that tuple (a, b) is in relation R, tuple (c, d) is in S, and b < c. Give a MapReduce implementation of this operation, assuming R and S are sets.

2.0.5 Answer:

```
Mapper(key null, (Set R, Set S)):
    for each tuple (a,b) in R:
        emit(1,(a,(b,R)))
    for each tuple (c,d) in S:
        emit(1,(d,(c,S)))

Reducer(key, list (record1, (record2, Set))):
    // record1 would be a or d.
    // record2 would be b or c.
    // Set would be R or S.
    for each pair in list:
        if pair.value.Set == R:
            emit((pair.record1, pair.value.record2), list)
            // this would emit((a,b),list)

Mapper(key (a,b), list (record1, (record2, Set))):
    for each pair in list:
        if (pair.value.Set == S) and (b > pair.value.record2):
            emit((a,b),(pair.value.record2,pair.record1))
            // this would emit((a,b),(c,d))

Reducer(key (a,b), list (c,d)):
```

```
for each pair in list:
    emit((a,b,c,d),(a,b,c,d))
```