Chuxin Zhang

Satish Kumar Thittamaranahalli

DSCI552

7 October 2020

Programming Assignment 3 Report

In this assignment, I used two algorithms, PCA, to deal with data from 3D to 2D and FastMap, to plot the words on a 2D plane. In this report I will detail my code structure and code-level optimizations performed.

START

```
import numpy as np
import matplotlib.pyplot as plt
import random as rd
import pandas as pd
```

For the libraries we used in this assignment, numpy is used to deal with array/matrix and some calculation; random is to generate random starting; matplot is used to make a plot in the FastMap part and panda is used to set a dataframe that can be plotted easily.

PCA

```
def minusMean(dt):
    mean = np.mean(dt,axis=0)
    newdt = dt-mean
    return newdt,mean
```

Firstly, we create a function to handle the dataset that contains x-mean(x), so that we can simply call it in the main function.

```
def main_pca(dt,n):
   newdt,mean = minusMean(dt)
   cov = np.cov(newdt,rowvar=0)
   eigval,eigvect = np.linalg.eig(cov)
   eigvalsort = np.argsort(-eigval)
   selectvec = np.matrix(eigvect.T[eigvalsort[:n]]).T
   fin = newdt * selectvec
   return selectvec,fin
```

In the main function, I found that numpy library has convenient function to get covariance and eigenvectors, eigenvalues (np.cov(), np.linalg.eig()). After we get eigenvalue, we use np.argsort() to get the indexes that sorting the eigenvalues from large to small. Then we choose the two largest eigenvalues from the indexes and calculate transform matrix. Final 2D data would be old data multiply direction. This function would return the direction and the final data.

```
pca_dt = np.genfromtxt('pca-data.txt',delimiter = '	')
pca_vec,pca_fin = main_pca(pca_dt,2)
```

After we defined the main function, we can execute the program by simply calling the function and specify its dimension.

PCA OUTPUT

PCA direction:

```
[[ 0.86667137 -0.4962773 ]

 [-0.23276482 -0.4924792 ]

 [ 0.44124968  0.71496368]]
```

PCA 2D dataset sample (I choose first two rows in the sample):

```
[[ 10.87667009   7.37396173]
```

```
 [-12.68609992  -4.24879151]]
```

FAST MAP

```
def findDist(a,b,count):
    if a == b:
        d = 0
    else:
        if count == 1:
            pointlst = [a,b]
            pointlst.sort()
            d = fm_dt[(fm_dt['a'] == pointlst[0]) & (fm_dt['b'] == pointlst[1])].iloc[0,2]
        else:
            d = (pow(findDist(a,b,count-1),2)-(pow(X[a][count-1]-X[b][count-1],2))) ** 0.5
    return d
```

In the FastMap, we firstly decide to create a function that can find distance between two objects. In the whole FastMap program, we referred a variable called ‘count’. It means the number of times in the regression. So, in the findDist function, if the count equal to 1, which means it is the first time of FastMap, thus the distance could directly take from fast-data.txt. If not, then we will calculate new distance from X.

Since we didn’t know the order of a and b object inputted in the function, and in the fast-data.txt we need to specify two objects from small to large, here I decided to use a sorted list to ensure the order of two object.

```
def findlocalFar(count):
    k = len(fm_wd)
    a = rd.randint(1,k)
    globalmax = -1
    far_a = 0
    far_b = 0
    localmax = 0
    while True:
        localmax = 0
        for i in range(1,k+1):
            d = findDist(a,i,count)
            if d > localmax:
```

```
            localmax = d
            far_a = a
            far_b = i
        if localmax > globalmax:
            globalmax = localmax
            a = far_b
        elif localmax <= globalmax:
            break
    pointlst = [far_a,far_b]
    pointlst.sort()
    far_a = pointlst[0]
    far_b = pointlst[1]
    return far_a,far_b
```

We then define a function to find two objects that have farthest distance. To realize this algorithm, I defined a local maximum and global maximum. If the local maximum is bigger than global maximum, the loop will continue and start at the destination object. If the global maximum is bigger, we can stop the loop. Before simply returning the farthest objects, I did another process to sort the point. I will explain this step later.

```
def FastMap(k):
    global count
    global X
    if (k<=0):
        return
    else:
        count+=1

    a,b = findlocalFar(count)
    length = len(fm_wd)
    for i in range(1,length+1):
        if i == a:
            X[i][count] = 0
        elif i ==b:
            X[i][count] = findDist(a,b,count)
        else:
            X[i][count]  = (pow(findDist(a,i,count),2) + pow(findDist(a,b,count),2) -
pow(findDist(b,i,count),2))/(2 * findDist(a,b,count))

    FastMap(k-1)
```

This is the main function. Firstly, we need to global count and X variable to make sure these two variables could work in the function. If k smaller or equal to 0, it means the algorithm has done, else, the count should be added. Ensuring the number of count, we then can run the main algorithm of FastMap. Finding the farthest objects, X could be defined in the for loop. At the end, we need to do a regression to calculate k-1 unless the k is equal to 0.

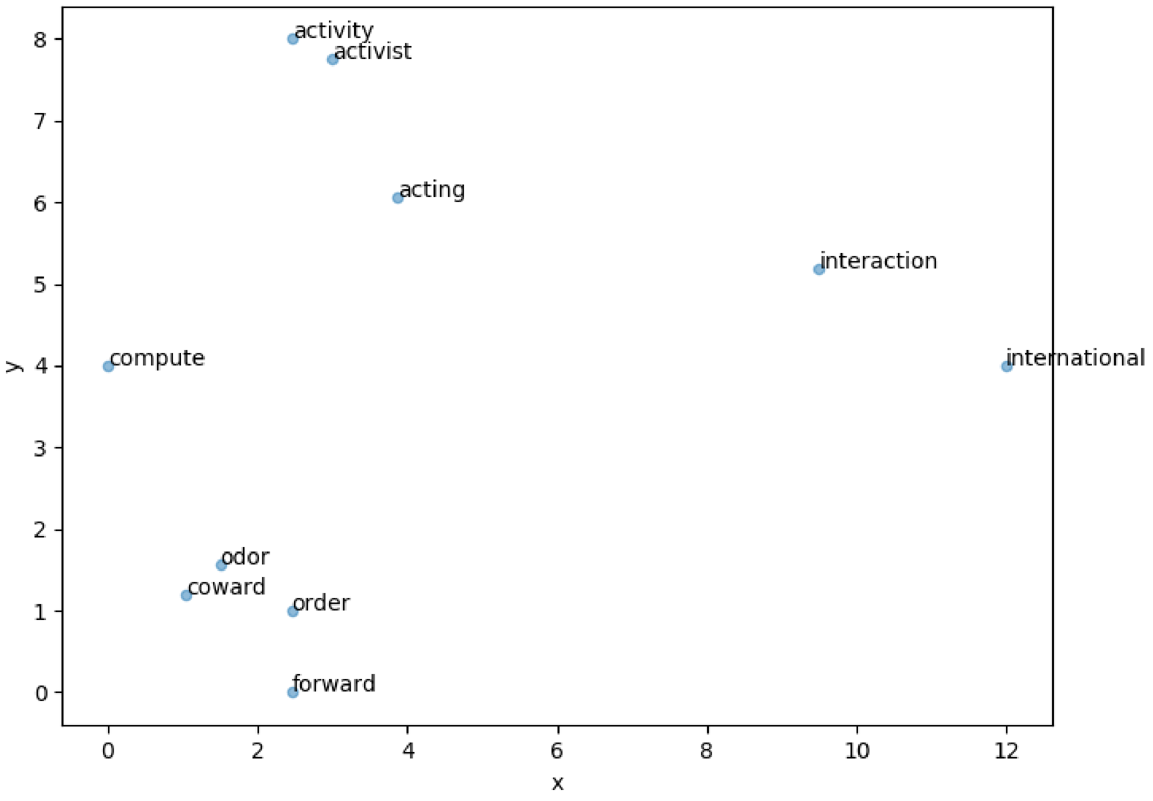
```
fm_dt = pd.read_table("fastmap-data.txt",sep='\t',names =['a','b','dis'])

with open('fastmap-wordlist.txt') as f:
    fm_wd = [line.rstrip() for line in f]

X = np.zeros([len(fm_wd)+1,2+1],dtype = float)
count = 0
FastMap(2)

###PLOT###
X = pd.DataFrame(X)
X.columns = ['word','x','y']
X = X.drop([0]).reset_index(drop=True)
X['word'] = pd.Series(fm_wd)

ax = X.plot.scatter(x='x', y='y', alpha=0.5)
for i, txt in enumerate(X.word):
    ax.annotate(txt, (X.x.iat[i],X.y.iat[i]))
plt.show()
```

In the execution part, I decided to input fastmap-data as dataframe from pandas. Then we can find the distance precisely. And fastmap-wordlist.txt has been inputted as a list. Before we execute the main function, the default of array X has been defined as len(objects) plus 1 and dimension we requested plus 1 for suiting index properly.

In the plot part, I firstly convert X we got to dataframe. Then we dropped the redundancy line and added the word object as another column. After the data being well defined, I used matplot to plot the graph.

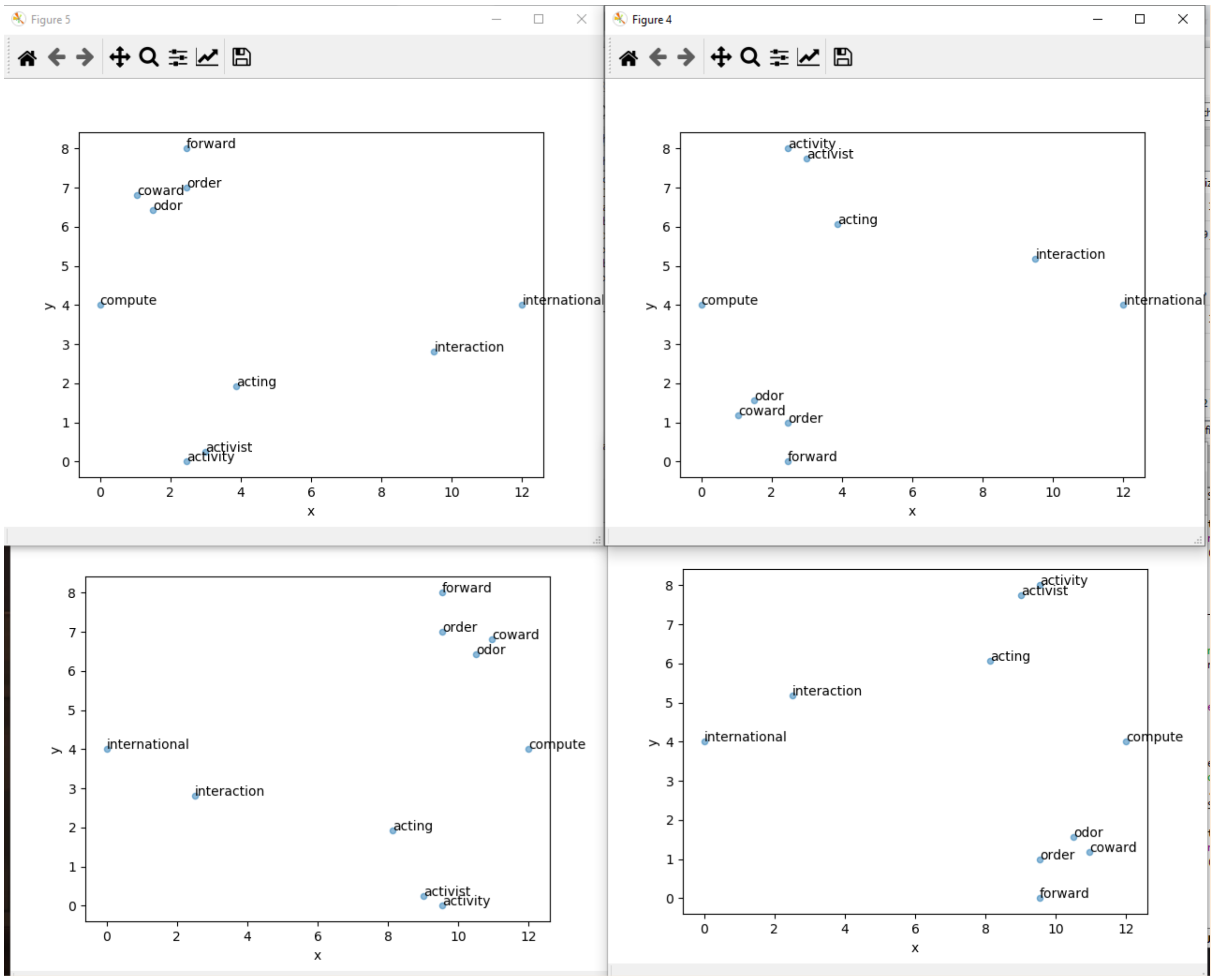
FASTMAP OUTPUT

```
            word          x       y
0         acting   3.875000  6.0625
1       activist   3.000000  7.7500
2        compute   0.000000  4.0000
3         coward   1.041667  1.1875
4        forward   2.458333  0.0000
5    interaction   9.500000  5.1875
6       activity   2.458333  8.0000
7           odor   1.500000  1.5625
8          order   2.458333  1.0000
9  international  12.000000  4.0000
```



Problem

FastMap initial value



When I firstly test the output of FastMap, there four different results but are central axial symmetry. After I re-reviewed the program, I noticed that it is because of farthest objects choosing. Since every time when starting at a random point, it might get an object with a front index, or probably with a latter index. For example, if the farthest objects is index 3 and index 10, the (a,b) could be (3,10) but could also be (10,3). This difference comes to be two results. And when regressed one more time, it will come to four results. That's why the results are central axial symmetry.

```
    pointlst = [far_a,far_b]
    pointlst.sort()
    far_a = pointlst[0]
    far_b = pointlst[1]
    return far_a,far_b
```

To fix the result, I added these steps when returning the farthest objects. After sorting the order in the list, the objects could be fixed from small index to large index.