INF552: Programming Assignment 3

Yi-Li Chen

Part 1: Implementation

PCA

- All imported library

```
1. import csv
2. import numpy as np
```

- Read text file into a list and convert to array

```
1. data = []
2. with open('pca-data.txt') as file:
3.     reader = csv.reader(file, delimiter='\t')
4.     for line in reader:
5.         data.append([float(line[0]),float(line[1]),float(line[2])])
6. data = np.array(data)
```

- Step0: Centralize the data point by deducting mean value

```
1. def data_centralize(data):
2.     mu = np.mean(data,axis=0)
3.     data_std = data-mu
4.     return data_std
```

- Step1: Calculate the covariance matrix for the features in the dataset.

```
1. def calculate_covariance(data_std):
2.     data_covar = np.cov(data_std, rowvar=False, bias=True)
3.     return data_covar
```

- Step2: Calculate the eigenvalues and eigenvectors for the covariance matrix.

```
1. def calculate_eigen(data_covar):
2.     eigenvalues,eigenvectors = np.linalg.eig(data_covar)
3.     return eigenvalues,eigenvectors
```

- Step3: Pick k eigenvalues and form a matrix of eigenvectors.

```
1. def build_k_dim_eigenvectorMatrix(eigenvectors, k):
2.     k_eigen = eigenvectors[:,0:k]
3.     return k_eigen
```

- After defining each step of algorithm, do the matrix multiplication of original data and matrix of eigenvectors with Nxk dimension.

```
def PCA(data, k):
    data_std = data_centralize(data)
    data_covar = calculate_covariance(data_std)
    eigenvalues,eigenvectors = calculate_eigen(data_covar)
    k_eigen = build_k_dim_eigenvectorMatrix(eigenvectors, k)
    print(k_eigen)
    new_data = np.matmul(data_std, k_eigen)
    return new_data
```

- The final result of PCA:

```
new_data = PCA(data, k=2)
```

The directions of the first two principal components:

1^{st} 2^{nd}

```
[[ 0.86667137 -0.4962773 ]
 [-0.23276482 -0.4924792 ]
 [ 0.44124968  0.71496368]]
```

FastMap

- All imported library

```
import csv
import random
import numpy as np
import matplotlib.pyplot as plt
```

- Read the fastmap_data into a list and convert to array

```
fastmap_data = []
with open('fastmap-data.txt') as file:
    reader = csv.reader(file, delimiter='\t')
    for line in reader:
        fastmap_data.append([float(line[0]),float(line[1]),float(line[2])])
fastmap_data = np.array(fastmap_data)
```

- Read the fastmap_wordlist into a list

```
fastmap_wordlist = []
with open('fastmap-wordlist.txt') as file:
    reader = csv.reader(file)
    for line in reader:
        fastmap_wordlist+=line
```

- Build a function that we can find out the corresponding farthest object when the one object and the distance are given

```
def find_farthest_obj(obj1_idx, fastmap_data):
    # The row related to obj1_idx
    obj1_distances = np.concatenate((fastmap_data[fastmap_data[:,0]==obj1_idx],
                                     fastmap_data[fastmap_data[:,1]==obj1_idx]))
    # Find out Oa in the above rows with max distance
    max_dist_idx = np.argmax(obj1_distances[:,2])
    obj2 = list(obj1_distances[max_dist_idx][0:2])
    obj2.remove(obj1_idx)
    obj2_idx = int(obj2[0])
    return obj2_idx
```

- Build a function that we can heuristically find out the pair of farthest objects:
  (1) Initialize an object arbitrarily and declare it to be Ob
  (2) Set Oa to be the object that is farthest apart from Ob according to the above function
  (3) Set Ob to be the object that is farthest apart from Oa again
  (4) Return the pair if Ob are same as the previous iteration

```
def find_farthest_pair(fastmap_wordlist, fastmap_data):
    Ob = random.sample(fastmap_wordlist,1)[0]
    Ob_idx = fastmap_wordlist.index(Ob)+1
    old_ob_idx = Ob_idx
    while 1:
        Oa_idx = find_farthest_obj(Ob_idx, fastmap_data)
        Ob_idx = find_farthest_obj(Oa_idx, fastmap_data)
        if old_ob_idx == Ob_idx:
            return Oa_idx, Ob_idx
        else:
            old_ob_idx = Ob_idx
```

- Build a distance function so that we can gain the distance when two objects and fastmap_data are given. This is because we can calculate the coordinate of object easily and more clearly.

```
def distance_function(obj1_idx, obj2_idx, fastmap_data):
    idx = []
    idx.append(obj1_idx)
    idx.append(obj2_idx)
    idx.sort()
    for idx_objs in range(len(fastmap_data)):
        if fastmap_data[:,0][idx_objs]==idx[0] and fastmap_data[:,1][idx_objs]==idx[1]:
            d = fastmap_data[:,2][idx_objs]
            return d
        else:
            d = 0
    return d
```

- In sum, firstly, initialize the Ob randomly, and find out the farthest pair of the fastmap data. Secondly, with the farthest pair of Oa and Ob, we can calculate the coordinate for each object. Then, update the fastmap with new distance function. Afterwards, we call FastMap function again with the new fastmap data and k-1. In addition, a stopping criteria which is when k=0 need to be set so as to return the and finish calculating all the coordinate for each object. One thing needs to be noticed is that the k will be subtracted 1 and the coordinate_idx will be added 1 for every next iteration.

```
def FastMap(k, fastmap_data, fastmap_wordlist, coordinate_idx):
    # Stopping criteria
    if k<=0:
        return
    else:
        coordinate_idx+=1
    # Find each farthest pair of fastmap_data
    Oa_idx, Ob_idx = find_farthest_pair(fastmap_wordlist, fastmap_data)
    # Calculate the coordinate for each iteration
    for obj_idx in range(1,len(fastmap_wordlist)+1):
        d_ai = distance_function(Oa_idx, obj_idx, fastmap_data)
        d_ab = distance_function(Oa_idx, Ob_idx, fastmap_data)
        d_ib = distance_function(Ob_idx, obj_idx, fastmap_data)
        x = (d_ai**2+d_ab**2-d_ib**2)/(2*d_ab)
        output_k_coordinate[obj_idx-1][coordinate_idx]=x
```

```
    # Create new distance function which is new fastmap data
    new_fastmap_data = fastmap_data.copy()
    for row in range(len(new_fastmap_data)):
        D_old = distance_function(new_fastmap_data[row][0],
                                  new_fastmap_data[row][1],
                                  fastmap_data)
        xi = output_k_coordinate[int(new_fastmap_data[row][0]-1)][coordinate_idx]
        xj = output_k_coordinate[int(new_fastmap_data[row][1]-1)][coordinate_idx]
        new_fastmap_data[row][2] = (D_old**2-(xi-xj)**2)**0.5
    # Call FastMap function again with the input of new fastmap
    FastMap(k-1, new_fastmap_data, fastmap_wordlist, coordinate_idx)
    return output_k_coordinate
```

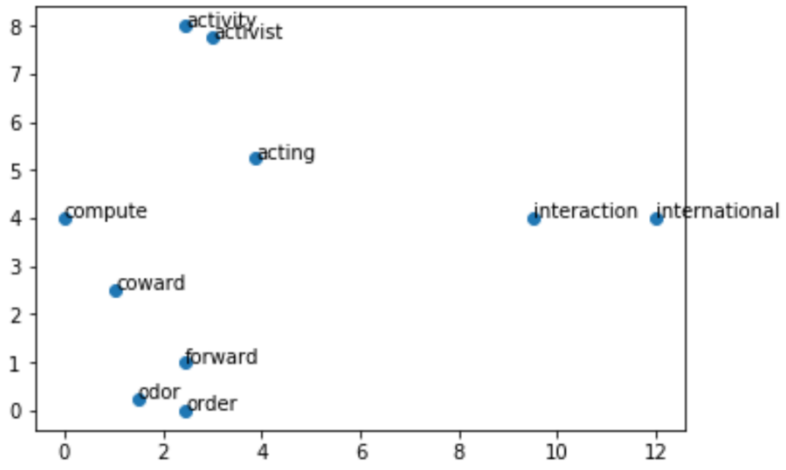
- The final result of FastMap algorithm:

```
output_k_coordinate = np.zeros((len(fastmap_wordlist),2)) # k=2
result = FastMap(2, fastmap_data, fastmap_wordlist, -1)
```

→ **The result of output coordinate**

```
array([[ 3.875     ,  5.25      ],
       [ 3.        ,  7.75      ],
       [ 0.        ,  4.        ],
       [ 1.04166667,  2.5       ],
       [ 2.45833333,  1.        ],
       [ 9.5       ,  4.        ],
       [ 2.45833333,  8.        ],
       [ 1.5       ,  0.25      ],
       [ 2.45833333,  0.        ],
       [12.        ,  4.        ]])
```

```
x = result[:,0]
y = result[:,1]
plt.plot(x,y,'o')
for xx,yy,word in zip(x,y,fastmap_wordlist):
    plt.annotate(word,(xx,yy))
```



In this assignment, the most challenging problem is to build the recursive algorithm for FastMap function. Since we have to make it to be a linear, we can't use two for-loop to run all the objects. We have to use one for loop to calculate the new distance function and the new fastmap data. Moreover, setting the stopping criteria is always the hardest things for me. I have to know clearly when it should stop and return the result. Last but not least, the relationship between the index of objects in fastmap-wordlist and the IDs in the fastmap-data need to be careful to deal with.