# INF 552 Assignment 3

PCA && Fastmap dimensionality reduction

Author:

Zongdi Xu(USC ID 5900-5757-70, working on PCA),

Wenkai Xu (USC ID 5417-1457-73, working on Fastmap)

Date: Feb 28, 2019

# Part 1

## (1) PCA Implementation

```
import numpy as np

# Read data from file
input_f = open('pca-data.txt', 'r')
data = input_f.readlines()
x = []
for record in data:
    x.append(record.split())
x = np.array(x).astype(np.float)
```

```
# Get the mean of input data in every dimension
n, dimension = x.shape
n=x.shape[0]
mean=np.sum(x,axis=0)/n
```

Output of “mean”:

```
array([ 0.04641608, -0.0356265 ,  0.06334316])
```

```
# Step 1&2: get the covariance matrix
cov_mat = (x - mean).T.dot((x - mean))/(n-1)
print(cov_mat)
```

Output of covariance matrix:

```
[[ 81.24199811 -15.84081415  31.66840483]
 [-15.84081415  13.70181418 -15.26445036]
 [ 31.66840483 -15.26445036  31.36677137]]
```

```
# Step 3: calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = np.linalg.eig(cov_mat)

print(eig_vecs)
print(eig_vals)
```

Output of eigenvectors and eigenvalues:

```
Eigenvectors
[[ 0.86667137 -0.4962773  -0.0508879 ]
 [-0.23276482 -0.4924792   0.83862076]
 [ 0.44124968  0.71496368  0.54233352]]

Eigenvalues
[101.61980038  19.89921519   4.79156808]
```

```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort()
eig_pairs.reverse()
```

Eigenvalues in descending order:

```
[(101.61980038291973, array([ 0.86667137, -0.23276482,  0.44124968])),
 (19.899215194176584, array([-0.4962773 , -0.4924792 ,  0.71496368])),
 (4.791568080870482, array([-0.0508879 ,  0.83862076,  0.54233352]))]
```

In this case, both eigenvalues of the 1st and 2nd dimensions are greater than the last, so these corresponding dimensions will be preserved after reduction.

```
# Step 4: apply projection to input data points
U = []
target_dimension=2
for i in range(target_dimension):
    U.append(eig_pairs[i][1])
U = np.array(U)
print U
```

Output of Matrix U:

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

```
result_x = np.dot(x, U.T)
print result_x
```

Result of PCA dimensionality reduction:

```
[[ 10.95314032   7.41375984]
 [-12.60962969  -4.2089934 ]
 [  0.50902129   0.30680664]
 ...
 [ -2.84606985   2.45894692]
 [ 11.25964147   4.24329087]
 [ 14.30637164   5.68389356]]
```

## 2）FastMap Implementation

Here is the brief heuristic of Fastmap algorithm:

1.We find the furthest distance points.

2.We make these 2 points' line as a dimension and then project all other points along the line.

We implement this by calling map() and distance() function recursively.

The detailed comments of my code are in the fastmap jupyter notebook file: fastmap.ipynb

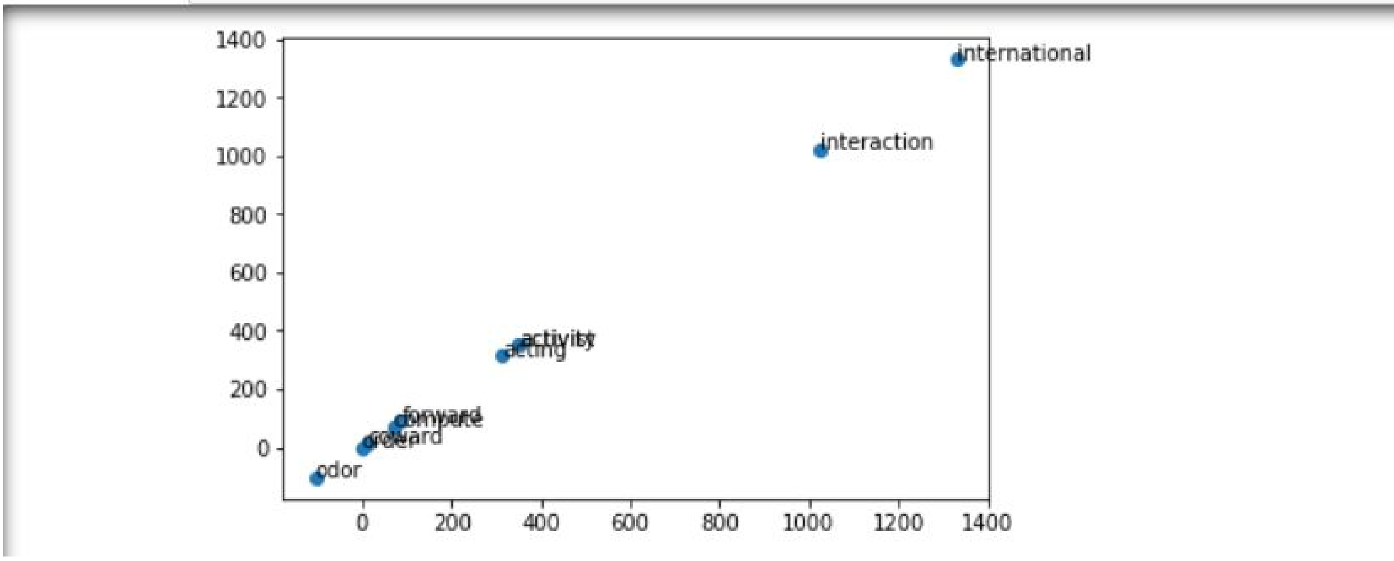
For the 1st part finding the distant points, I use to for loop to find the most distant points. Here's the code:

```
localmax=-111111
for x in range (0,len(self.M)-1):
# For this part, we are going to find the furthest distance
    for y in range (x+1,len(self.M)):
        if(self.Distance(x,y, self.column)>localmax):
            localmax=self.Distance(x,y,self.column)
            p1=x
            p2=y
self.point[self.column]=(p1,p2)
# p1,p2 are the coordinates of both points
```

For the 2nd part of projection, I call the function of map and distance recursively to project all other points along the "furthest line". Here is the code:

```
if self.Distance(p1,p2,self.column)==0:
    return
for i in range(len(self.M)):
# Here is the projection of the ith point on to the chose points
    dix=self.Distance(i,x,self.column)
    diy=self.Distance(i,y,self.column)
    dxy=self.Distance(x,y,self.column)

    self.r[i][self.column]=(dix + dxy - diy) / 2*math.sqrt(dxy)

self.column=self.column + 1
self.map(D-1)

def Distance(self,x,y,col):
# Distance function is to compute the distances based on projection before
    if col==0: return self.M[x,y]**2

    m=self.Distance(x,y,col-1)
    n=(self.r[x][col] - self.r[y][col])**2
    return m-n
```

Here is the plot of the words on the 2-D spaces:




# Part 2

## (1) PCA Software Familiarization

Here we use library function from “scikit-learn”, a machine learning library in python.

```
import numpy as np
from sklearn.decomposition import PCA

input_f = open('pca-data.txt', 'r')
data = input_f.readlines()
x = []
for record in data:
    x.append(record.split())
x = np.array(x).astype(np.float)
original_x = x.copy()

n, dimension = x.shape
target_dimension = 2
pca = PCA(n_components=target_dimension)
print pca.fit_transform(x)
```

Result of PCA dimensionality reduction:

```
[[-10.87667009   7.37396173]
 [ 12.68609992  -4.24879151]
 [ -0.43255106   0.26700852]
 ...
```

```
[  2.92254009   2.41914881]
[-11.18317124   4.20349275]
[-14.2299014    5.64409544]]
```

### 2) FastMap Software Familiarization

This is a package I can find from the internet:

**Fastmap Dimensionality reduction by Christos Faloutsos**

To compile the program

**> make main**

can simply run by

**> main <filename>**

can have few options

**> main [-v] [-d <num>] <filename>**

**where**

**-v give the verbose option**

**-d set the dimension number, if no input then default dimension=2**

in the file provided, can accept one of four types of data

**objects(like vectors)**

**object arrays**

**vectors with length**

**2d matrices with length**

any of these data can be accepted as the data for forming fast map

# Part 3: Application

Generally speaking, data preprocessing is one of the non-negligible steps before neural network training.  Usually original data contains too much attributes and variation to deal with.  In this case, we have to filter out some more significant factors.  Dimensionality reduction is just such a method to underline those factors that are worthwhile being paid attention to.

Here is what we might improve in our codes:
In the fastmap algorithm, I use two loops to find the longest distances. It has the time complexity of O(N^2). It is kind of time consuming and we might find a better way to solve it.

# References:


1.https://github.com/mahmoudimus/pyfastmap/blob/master/stringmap.py#L135
2.http://gromgull.net/2009/08/fastmap.py