HOMEWORK #3
Principal Component Analysis and FastMap

Group Members

Name	USC ID
Aayush Sinha	7268779355
Abhishek Jangalwa	8882321574
Radhika Agarwal	3865981998

Table of Contents

PYTHON VERSION

- All the programs are written in **Python 2**

COMPARISON WITH SCIKIT-LEARN

Scikit learn has “PCA” class under “decomposition” collection. It uses Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. It is capable of returning many useful information about the data set once we call the function “fit_transform()”. The comparison with our code is as follows:

1) sklearn.decomposition.PCA has multiple parameters to tune the algorithm such as svd_solver, iterated_power and random_state. Svd_solver can be set to multiple values like auto, full, arpack and radndomize, each tweaking the algo.

2) Sklearn can also give us the explained_variance_ratio which is percentage of variance explained by each of the selected components.

3) Sklearn can return noise_variance which is the estimated noise covariance following the Probabilistic PCA model from Tipping and Bishop 1999 (http://www.miketipping.com/papers/met-mppca.pdf). It is required to computed the estimated data covariance and score samples. It is a floating point value.

APPLICATIONS OF FASTMAP

1. The FastMap Algorithm for Shortest Path Computations **https://www.researchgate.net/publication/317543418_The_FastMap_Algorithm_for_Shortest_Path_Computations**

A pre-processing algorithm for embedding the nodes of a given edge-weighted undirected graph into a Euclidean space. At runtime, a shortest path between any two nodes can be computed using A* search with the Euclidean distances as heuristic estimates. The pre-processing algorithm, dubbed FastMap, runs in near-linear time. Hence, FastMap is orders of magnitude faster than competing approaches that produce a Euclidean embedding using Semi definite Programming. This FastMap algorithm also produces admissible and consistent heuristics and therefore guarantees the generation of optimal paths.

2. FASTMAP software package for reducing magnetic field imperfections in magnetic resonance imaging and spectroscopy

http://innovation.columbia.edu/technologies/cu17327_fastmap-a-user-friendly-software-package-for-efficient-b0-shimming-in-magnetic-resonance

FASTMAP method introduced by Gruetter et al is used to derive B0 field information along six selected projections, producing significant time savings compared with B0 shimming techniques that require full 3-dimensional imaging.

3. A Visual Method of Cluster Validation with Fastmap
https://link.springer.com/chapter/10.1007/3-540-45571-X_18

In the visual cluster validation with FastMap, clusters are first generated with a clustering algorithm from a database. Then, the FastMap algorithm is used to project the clusters onto a 2-dimensional (2D) or 3-dimensional (3D) space and the clusters are visualized with different colors and/or symbols on a 2D (or 3D) display. From the display a human can visually examine the separation of clusters. This method follows the principle that *if a cluster is separate from others in the projected 2D (or 3D) space, it is also separate from others in the original high dimensional space* (the opposite is not true). The modified FastMap algorithm improves the quality of visual cluster validation by optimizing the separation of clusters on the 2D or (3D) space in the selection of pivot objects (or projection axis).

4. Spatio-temporal fastmap-based mapping for human action recognition

http://ieeexplore.ieee.org/document/7532919/
Automatic recognition of human actions in video is useful for surveillance, content-based summarization and human-computer interaction applications, to name a few. However, it is also a very complex task due to viewpoint variations, occlusions, background interference, movement variability of the same action and ambiguity between different actions. Thus applying a short temporal set of FastMap dimensionality reduction-based technique for embedding a sequence of raw moving silhouettes, associated to an action video into a low-dimensional space, in order to characterize the spatio-temporal property of the action, as well as to preserve much of the geometric structure.

APPLICATIONS OF PCA

1) Detection and Visualization of Computer Network Attacks:

(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.9147&rep=rep1&type=pdf)

Network traffic data collected for intrusion analysis is typically high-dimensional, making it difficult to both analyze and visualize. Principal Component Analysis is used to reduce the dimensionality of the feature vectors extracted from the data to enable simpler analysis and visualization of the traffic. Visualization of network activity and possible intrusions is achieved using Bi-plots, which provides a summary of the statistics.

2) Ranking Players of Cricket (Sport):

(http://ww2.amstat.org/publications/jse/v21n3/scariano.pdf)

Principal component analysis is successfully used to rank the cricket batsmen and bowlers who played in the Indian Premier League (IPL) competition. The first principal component, like the total runs scored, is seen to explain a substantial portion of the variation in a linear combination of some commonly used measures of cricket prowess. Performance measures can be used to decide a particular player's price in future auctions, and there are several indicators typically used to measure the performances of players. Eg. when considering batsmen, the goal in limited-overs cricket, like Twenty20, is to score as many as runs as possible using as few balls as possible.

3) Distinguish Patients with Schizophrenia from Healthy Controls Based on Fractional Anisotropy Measurements:

(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2566788/)

Researchers measured the distance between two populations using Mahalanobis distance and chose the eigenvectors to maximize it, a modified PCA method, which they call the discriminant PCA (DPCA). DPCA was applied to diffusion tensor based fractional anisotropy images to distinguish age-matched schizophrenia subjects from healthy controls. The resulting classification came out to be at minimum error.

4) Sensory assessment tool for fermented food products:

(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3614048/)

The objective of the work was to use the method of quantitative descriptive analysis (QDA) to describe the sensory attributes of the fermented food products prepared with the incorporation of lactic cultures. Panelists were selected and trained to evaluate various attributes specially color and appearance, body texture, flavor, overall acceptability and acidity of the fermented food products like cow milk curd and soymilk curd, idli, sauerkraut and ice cream. Principal component analysis (PCA) identified the six significant principal components that accounted for more than 90% of the variance in the sensory attribute data. Overall product quality was modelled as a function of principal components using multiple least squares regression ($R^2 = 0.8$). The result from PCA was statistically analyzed by analysis of variance (ANOVA). These findings demonstrate the utility of quantitative descriptive analysis for identifying and measuring the fermented food product attributes that are important for consumer acceptability.

Data Structures

Pandas DataFrames:

Pandas is a library in python providing easy-to-use data structures for data analysis. DataFrame is a 2-dimensional labelled data structure that contains columns with different data types. A DataFrame can be created from:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

DataFrames provide a very convenient method to visualize the raw data as a table and use various in built functions. Using these methods, it allows us to write a highly optimized code and the implementation also becomes easier.

```
        ...
data=pd.read_table(location)
table=pd.DataFrame.as_matrix(data)
        ...
```

Numpy Array Structure –
Used Numpy matrix to store the covariance matrix. Numpy Matrix allows matrix operations like Transpose, Covariance of a matrix, finding the Eigen values and Eigen Vectors.

```
        ...
covariance_matrix=np.cov(table.T)
e_vals,e_vects=np.linalg.eig(covariance_matrix)
reduced=np.dot(table,result)
        ...
```

Challenges Faced

- Verification of the result in PCA was a challenge.
- We have used np.linalg.eig to calculate the eigen values and eigen vectors. Manually calculating it would have been a major challenge.
- In FastMap, while computing the cosine distances for an object the values evaluated were NaN initially when the formula was multiplied by 0.5 instead of division by 2. After changing it to division by 2 issue was resolved.

Optimizations

- Used Numpy.array, which provides a much faster implementation of lists. Also, calculations such as eigen vectors and values become easier to calculate if data is stored in a Numpy array.
- Used argsort to sort the eigen values to get the index.
- Created a class FMap with variables n (number of dimensions) and newDimension(Final reduced dimension of the given data set), thus providing ease of access across all the functions.
- Used max() function to find the maximum number in a row in farthest_point() function instead of traversing through the row

Improvements in Code

- Runtime can be decreased.

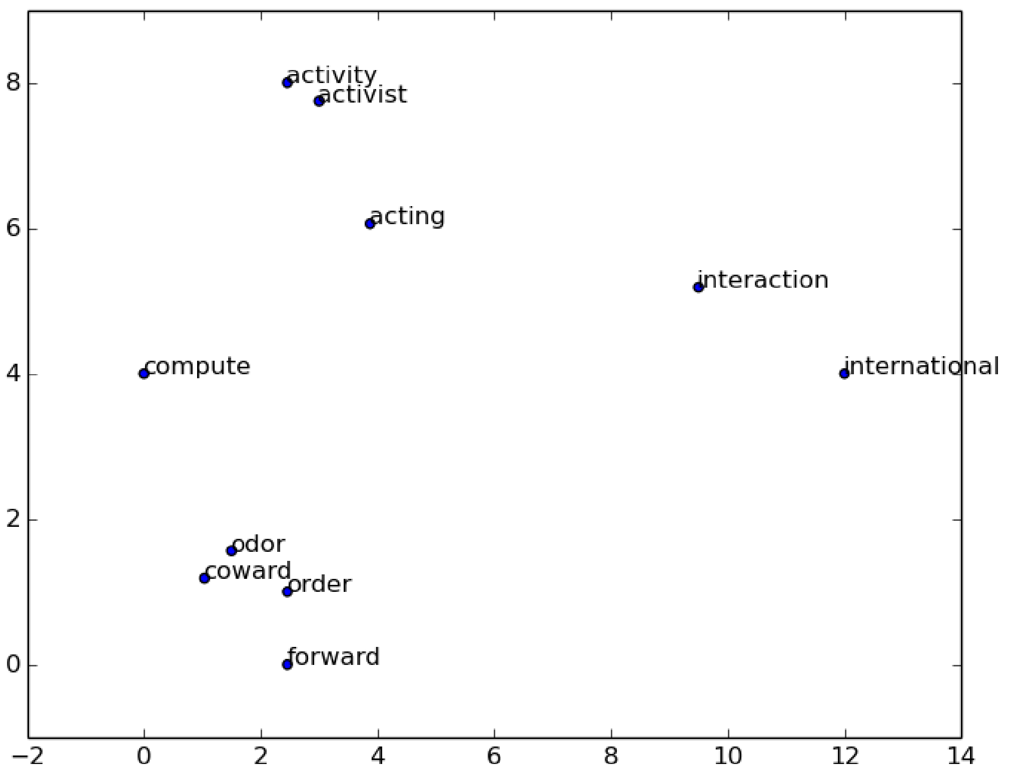
Output

PCA:

```
[[-0.86675083  0.23265555 -0.44115121]
 [-0.49614303 -0.49239263  0.71511649]]
```

FastMap:

```
[[  3.875        6.0625    ]
 [  3.           7.75      ]
 [  0.           4.        ]
 [  1.04166667   1.1875    ]
 [  2.45833333   0.        ]
 [  9.5          5.1875    ]
 [  2.45833333   8.        ]
 [  1.5          1.5625    ]
 [  2.45833333   1.        ]
 [ 12.           4.        ]]
```



Explanation of the code

Principal Component Analysis

- The input data is taken and stored in an Numpy Matrix.
- The covariance of the input data is calculated.
- The Eigen values and the Eigen Vectors of the covariance matrix are calculated.
- The Eigen Values are sorted in decreasing order and the corresponding Eigen Vectors give the direction of the principal components. The Eigen vectors corresponding to the lower Eigen values do not possess valuable information and hence they are discarded.
- The do product of the Eigen vectors and the original matrix gives the reduced data set.

FastMap

- Train – Reads the data and creates a N X N matrix which stores the pair-wise distance of the objects.
- Farthest_point – It computes the set of two farthest located objects, it randomly starts with an object (Oa), then find the object located farthest away from it (Ob). Then find the object located farthest from Ob which is Oc. If Oa equals Oc then the farthest pair is found else Ob becomes Oa and Oc becomes Ob, and above process is repeated until Oa equals Oc. Usually we find the pair in 2 iterations.

- fastMap – This function implements the fastMap algorithm. It takes as an input number of reduced dimensions and the distance matrix. First it calls the farthest_point function to compute the points located farthest, then it finds the cosine distance of every other objects with respect to the line between Oa and Ob. After the new co-ordinate is found the new distance matrix is evaluated where distance between Objects are calculated on a hyperplane. New Distance = Original Distance – difference between projections. Once new matrix is calculated it is recursively passed to the fastMap function along with the new dimension numbers to be found i.e. (n-1). Process repeats until all the dimensions are evaluated.

Python Code

PCA:

```
# -*- coding: utf-8 -*-

# Implementation of PCA
# code written by:
# Aayush Sinha
# Abhishek Jangalwa
# Radhika Agarwal


import numpy as np
import pandas as pd
import sys

location = sys.argv[1]
data=pd.read_table(location)
table=pd.DataFrame.as_matrix(data)

#calculating covariance manually
mean_val=np.mean(table,axis=0)
sub=table-mean_val
covariance_matrix = (sub).T.dot((sub)) / (len(sub)-1)

#Eigen vectors and Eigen values
e_vals,e_vects=np.linalg.eig(covariance_matrix)

#Finding Eigen vectors based on Max eigen values:
val=(-e_vals).argsort()[:2]
result=e_vects[...,val]
print result.T

# data modified into a lower dimension
reduced=np.dot(table,result)
#print "modified data set\n",reduced
```

FastMap

```
# -*- coding: utf-8 -*-

# Implementation of Fast Map
# code written by:
```

```
# Aayush Sinha
# Abhishek Jangalwa
# Radhika Agarwal

import numpy as np
import pandas as pd
import sys
import matplotlib.pyplot as plt


class FMap:

    def __init__(self, num_dimension, newDimension = []):
        self.num_dimension = num_dimension
        self.newDimension = newDimension

    def train(self,data):
        data = data.values
        dist_array = np.zeros((10, 10), dtype = np.int)

        for item in data:
            i = item[0]
            j = item[1]
            dist_array[i-1][j-1] = item[2]
            dist_array[j-1][i-1] = item[2]

        self.fastMap(self.num_dimension, dist_array)

    def farthest_point(self,distance):

        obj_a = np.random.randint(0,9)
        while True:
            farthest = max(distance[obj_a])
            obj_b = distance[obj_a].index(farthest)
            tmpDistance = max(distance[obj_b])
            tmpObj = distance[obj_b].index(tmpDistance)
            if (tmpObj == obj_a):
                break
            else:
                obj_a = obj_b

        if obj_a < obj_b:
            return (obj_a, obj_b)
        else:
            return (obj_b, obj_a)


    def fastMap(self, n, distance):
        if n<=0:
            return
        distance = distance.tolist()
        pivots = self.farthest_point(distance)

        a = pivots[0]
        b = pivots[1]
        farthest = distance[a][b]
        dimension = []

        for i in range(10):
            tmpDistance = 0
            if i == a:
                dimension.append(0)
            elif i == b:
                dimension.append(farthest)
            else:
                tmpDistance = ((distance[a][i]**2) + (farthest**2) - (distance[b][i]**2))/float(2 *
farthest)
                dimension.append(tmpDistance)

        self.newDimension.append(dimension)
        projection = np.zeros((10, 10))

        if (n >= 1):
            for i in range (10):
                for j in range (10):
                    tmp = (distance[i][j] ** 2) - ((dimension[i] - dimension[j]) ** 2)
                    projection[i][j] = np.sqrt(tmp)
            self.fastMap(n-1, projection)

def main():
    #readfile = r"C:\Users\amiya\Desktop\USC GRAD\Machine
Learning\Assignment\HW3\fastmap-data.txt"
    #nameFile = r"C:\Users\amiya\Desktop\USC GRAD\Machine
Learning\Assignment\HW3\fastmap-wordlist.txt"

    readfile = sys.argv[1]
    nameFile = sys.argv[2]
    fd=open(nameFile,"r")

    name = fd.read()
    nameList = name.strip().split('\r\n')

    data = pd.read_csv(readfile, sep = '\t', header = None)

    num_dimension = 2

    FastMap = FMap(num_dimension)
    FastMap.train(data)

    newDimension1 = np.array(FastMap.newDimension)
    newDimension1=newDimension1.T

    print newDimension1


    fig,ax = plt.subplots()

    ax.scatter(newDimension1[:,0], newDimension1[:,1])

    for i, txt in enumerate(nameList):
        ax.annotate(txt, (newDimension1[i][0], newDimension1[i][1]))

    plt.show()

if __name__ == '__main__':
    main()
```

Contributions

Code PCA: Aayush, Abhishek
Code FastMap: Radhika, Aayush
Scikit-Learn: Abhishek
Report: Aayush, Abhishek, Radhika