INF-552
MACHINE LEARNING FOR DATA SCIENCE

PROGRAMMING ASSIGNMENT 2:
PCA & FASTMAP

NAME	USC-ID	EMAIL
Amitabh Rajkumar Saini	7972003272	amitabhr@usc.edu
Shilpa Jain	4569044628	shilpaj@usc.edu
Sushumna Khandelwal	7458911214	sushumna@usc.edu

PART 1: IMPLEMENTATION

- **Data Structures Defined/Used**
  - Dataset represented as numpy 2-D Array
  - Dimensionality Reduction from 3D to 2D

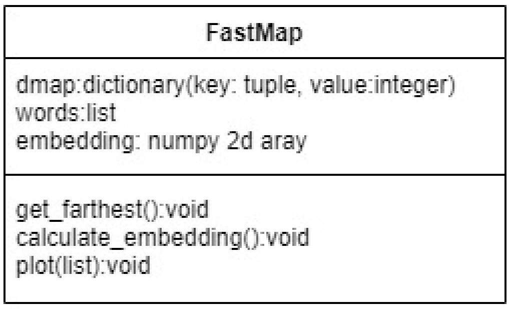
Principal Component Analysis

In the implementation of PCA, reduce the dimensionality of data points from 3D to 2D, and the output will be the directions of first two principal components.

1. According to the data set and given number of dimension, assign 2 to goal dimension that want to reduce to.
2. Get all data points from input file, pca-data.txt, into system. The data file contains 6000 points of 3-dimension data points.
3. Normalize the data point:
   a. Calculate the mean value μ of all data points with numpy package.
   b. Replace data point X(i) with X(i) – μ.
4. Calculate the covariance matrix with numpy package.
5. Calculate the eigenvectors of covariance:
   a. Calculate the original eigenvectors with numpy package.
   b. According to eigenvalue, sort the eigenvectors.
   c. Return the first 2 columns of eigenvectors _v_k.
6. Make projections on 2D space and get 2D representation of data points.
   a. Multiply the _v_k's transposition and X's transposition, and then transpose the result as the final representation of data points.
7. Plot the new data points on 2D space with matplotlib.pyplot package

Fastmap

In the implementation of FastMap, map the data points to a 2D space just based on the symmetric distance between them. The output will be the distances between the projections on two dimensionalities.



- **Code: (Language - Python)**

Principal Component Analysis:

```python
'''

PA-3: Principal Component Analysis
Authors:
Amitabh Rajkumar Saini, amitabhr@usc.edu
Shilpa Jain, shilpaj@usc.edu
Sushumna Khandelwal, sushumna@usc.edu

Dependencies:
1. numpy : pip install numpy
2. matplotlib : pip install matplotlib
3. mplot3d : pip install mplot3d

Output:
Returns a 2D transformed data, writes model eigen values and transformed
data on console and generates the plot of the same
(it takes time to generate 2D plot and 3d plot)

'''

import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt #Used for plotting graph
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    '''
    class to display eigen vectors on 3D plane
    '''
    def __init__(self, xs, ys, zs, *args, **kwargs):
        '''
        :param xs: mean of x,eigen value
        :param ys: mean of y,eigen value
        :param zs: mean of z,eigen value
        '''
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        '''

        :param renderer: to render points
        setting values of plot
        :return: returns nothing
        '''
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

def plot_eigen_vectors_graph(all_samples, eig_vec_sc, mean_x, mean_y,
mean_z):
    '''
    Displays a graph containing 3 eigen vectors
    :param all_samples: data matrix numpy array
    :param eig_vec_sc: eigen vectors numpy array
    :param mean_x: mean of column1
    :param mean_y: mean of column2
    :param mean_z: mean of column 3
    :return:  nothing
    '''
    fig = plt.figure(figsize=(7,7))
    ax = fig.add_subplot(111, projection='3d')

    ax.plot(all_samples[0,:], all_samples[1,:], all_samples[2,:], 'o',
markersize=8, color='green', alpha=0.2)
    ax.plot([mean_x], [mean_y], [mean_z], 'o', markersize=10,
color='red', alpha=0.5)
    for v in eig_vec_sc.T:
        a = Arrow3D([mean_x, v[0]], [mean_y, v[1]], [mean_z, v[2]],
mutation_scale=20, lw=3, arrowstyle="-|>", color="r")
        ax.add_artist(a)
    ax.set_xlabel('x_values')
    ax.set_ylabel('y_values')
    ax.set_zlabel('z_values')

    plt.title('Eigenvectors')

    plt.show()

def plot3D(X):
    '''
    Displays a plot depicting 3d points on graph
    Return nothing
    '''
    x_points = [each[0] for each in X]
    y_points = [each[1] for each in X]
    z_points = [each[2] for each in X]
    fig = plt.figure(figsize=(8, 8))
    ax = plt.axes(projection="3d")
    ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');
    plt.legend('3D Plot')

    plt.show()

def plot2D(reduced_data):
    '''
    Displays a plot depicting 2d points on graph
    Return nothing
    '''

    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    for i in range(len(reduced_data)):
        _x = reduced_data[i][0]
        _y = reduced_data[i][1]

        plt.plot(_x, _y, 'y.', markersize=1)

    plt.show()
```

```
def pca(data_file,k):
    '''
    Runner Program
    :return: returns nothing
    '''

    #Data loaded in numpy array
    data=np.loadtxt(data_file,delimiter="\t")
    data=data-np.mean(data,axis=0)

    #numpy function to calculate covariance matrix
    short_cov = np.cov(data.T)

    #Eigen value and eigen vector calculation
    e_values , e_vectors = np.linalg.eig(short_cov)

    mean_val = np.mean(data, axis=0)
    mean_x, mean_y, mean_z = mean_val[0], mean_val[1], mean_val[2]

    plot_eigen_vectors_graph(data, e_vectors, mean_x, mean_y, mean_z)

    #Merging eigen value and eigen vector and sorting it into descending
order.
    e_values_sorted = sorted(list(zip(e_values, e_vectors)), key=lambda
x: x[0], reverse=True)
    print("Eigen Vectors")
    for i in e_vectors:
        print(i)
    #print(e_vectors)
    print("\n")
    print("Eigen Values")
    for i in e_values:
        print(i)

    #Picking k columns
    u_matrix = np.empty((data.shape[1], k))
    for i in range(len(e_values_sorted)):
        u_matrix[i] = e_values_sorted[i][1][:k]

    #Transformed 2d matrix
    z = np.dot(u_matrix.T, data.T)
    print("\n")
    print("Transformed Data into 2D")
    print(z.T)

    #Graph plot
    plot3D(data)
    plot2D(z.T)


#  print(e)
if __name__ == "__main__":
    pca('pca_data.txt',2)
```

FastMAP:

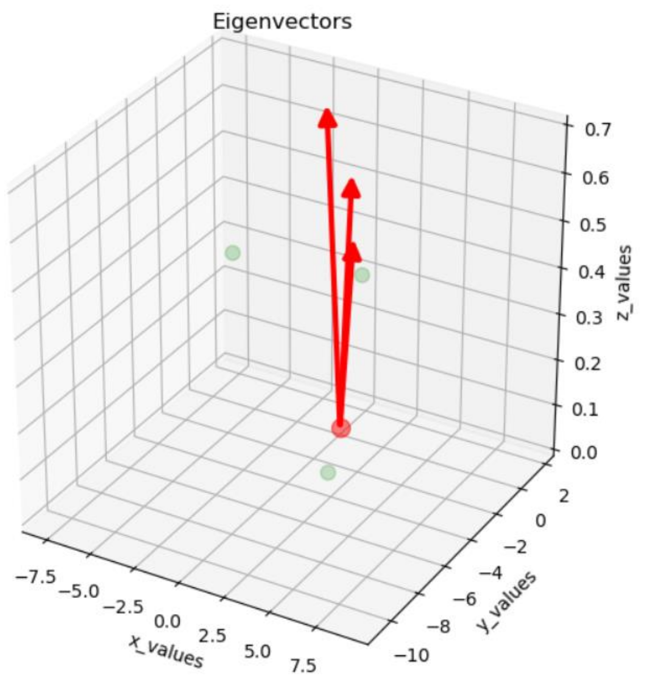
```
    '''

PA-3: FastMap
Authors:
Amitabh Rajkumar Saini, amitabhr@usc.edu
Shilpa Jain, shilpaj@usc.edu
Sushumna Khandelwal, sushumna@usc.edu

Dependencies:
1. numpy : pip install numpy
2. matplotlib : pip install matplotlib

Output:
Return the mapped the data points to a 2D space just based on the
symmetric distance between them.
The output will be the distances between the projections on two
dimensionalities and a plot graph for the same


'''



import random
import numpy as np
import math
import matplotlib.pyplot as plt

class fastmap:
    def __init__(self,dmap,word,k):
        '''
        Constructor called when object  is invoked
        :param dmap: dictionary for storing fastmap-data with key as
tuple indicationg the word and value as distance between those two
words
        :param word:word list
        :param k:dimension
        :param embedding: Final output

        '''

        self.dmap=dmap   #(x,y):dist
        self.words=word #words
        self.k = k
        self.embedding = np.zeros((len(word)+1,self.k))
#[[x0,y0][x2,y2],[x3,y3]..]

    def get_farthest(self):
        '''
        Calculates the farthest points by not traversing all the
points as discussed in class
        :return the two farthest points in the form of tuple
        '''
        rand_index = random.randint(0,len(self.dmap)-1)
        pivot = 
random.choice([list(self.dmap.keys())[rand_index][0], 
list(self.dmap.keys())[rand_index][1]])
        #print(pivot)
        prev = None
        while(True):
            maxi = float('-inf')
            next = None
            for i in range(1,len(self.words)+1):
                if i == pivot:
                    continue
                if (i,pivot) in self.dmap:
                    d=self.dmap[(i,pivot)]
                else:
                    d=self.dmap[(pivot,i)]

                if maxi < d:
                    maxi = d
                    next = i

            if prev == next:
                break

            prev = pivot
            pivot = next

        return (pivot,next)

    def calculate_embedding(self):
        '''
            Embeds the points on hyperplane for k =2 each time and 
updates the distance by subtracting the distnace calculated in 
            previous iteration
            :returns the funcyion returns nothing
        '''

        k = 0
        #print(self.dmap)
        while k < self.k :
            far = self.get_farthest()
            if far not in self.dmap:
                far = (far[1],far[0])
            oa , ob = far
            for i in range(1,len(self.words)+1):
                if i == oa:
                    self.embedding[i][k]=0
                    continue
                elif i == ob:
                    self.embedding[i][k]=self.dmap[far]
                    continue
                pta = (i,oa) if (i,oa) in self.dmap else (oa,i)
                ptb = (i,ob) if (i,ob) in self.dmap else (ob,i)
                di = (self.dmap[far]**2 + self.dmap[pta]**2 - 
self.dmap[ptb]**2)/(2*self.dmap[far])
                self.embedding[i][k] = di
            self.update_dist(k)

            k+=1
    def plot(self, label_set):
        '''

        :param label_set:list containing word list
        :return: the function returns nothing just plots a graph
representing the word list on 2d plane
        '''
        plt.xlabel("x-axis")
        plt.ylabel("y-axis")
        self.embedding=self.embedding[1:]

        for i in range(1,len(self.embedding)):
            _x = self.embedding[i][0]
            _y = self.embedding[i][1]
            _label = label_set[i]
            plt.plot(_x, _y, 'b.', markersize=10)
            plt.annotate(
                _label, xy = (_x, _y), xytext = (30, 20), textcoords
= 'offset points', ha = 'right', va = 'bottom',
                bbox = dict(boxstyle = 'round,pad=0.5', fc =
'yellow', alpha = 0.1),
                arrowprops = dict(arrowstyle = '->', connectionstyle
= 'arc3,rad=0'))
        plt.show()


    def update_dist(self,k):
        '''
        :param k: denoting the column in embedding
        :return: the updated distance by subtarcting teh distance in
previous iteration
        '''
        for each in self.dmap.keys(): #(2,3)
            self.dmap[each] = math.sqrt((self.dmap[each]**2 -
(self.embedding[each[0]][k]-self.embedding[each[1]][k])**2))

    def map_words(self):
        '''
        Combines the words and coordinates
        :return: the function returns nothing just prints the o/p on
console
        '''
        map_words = zip(self.words,self.embedding[1:,:])
        #print(list(map_words))
        print("Word\t\tEmbedding")
        for each in map_words:
            print(each[0]+"\t\t"+str(each[1]))


def main():
    '''
    Runner Program
    :return: returns nothing
    '''
    d_map={}
    #Data loaded into a numpy array
    data=np.loadtxt('fastmap-data.txt')
    #Word-list loaded in list
    words=open('fastmap-wordlist.txt')
    words=words.read().split('\n')
    #k : dimension
    k = 2
    for i in data:
        d_map[(int(i[0]),int(i[1]))]=int(i[2])
    #object instantiation
    fmap = fastmap(d_map,words,k)
    fmap.calculate_embedding()
    fmap.map_words()
    fmap.plot(words)


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

- **Run and Output the result and graph of transformed 2d data**

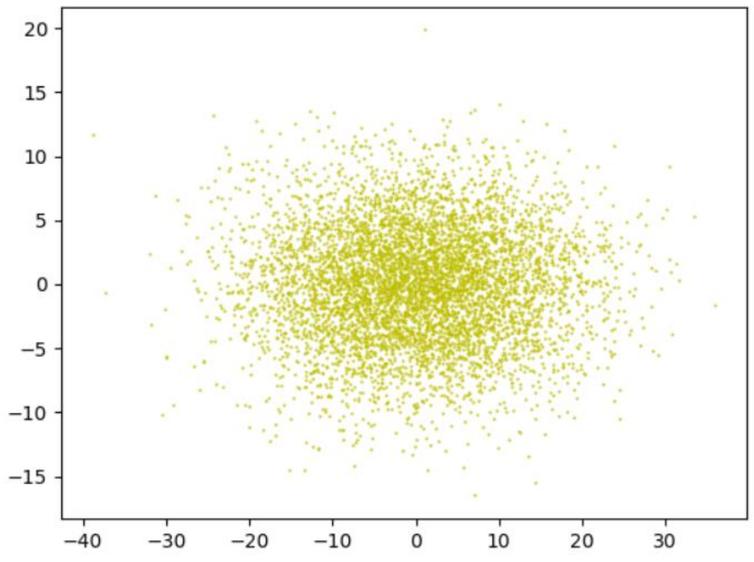
PCA

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


Eigen Values
101.6198003829197
19.89921519417658
4.791568080870478


Transformed Data into 2D
[[ 10.95314032   7.41375984]
 [-12.60962969  -4.2089934 ]
 [  0.50902129   0.30680664]
 ...
 [ -2.84606985   2.45894692]
 [ 11.25964147   4.24329087]
 [ 14.30637164   5.68389356]]
```

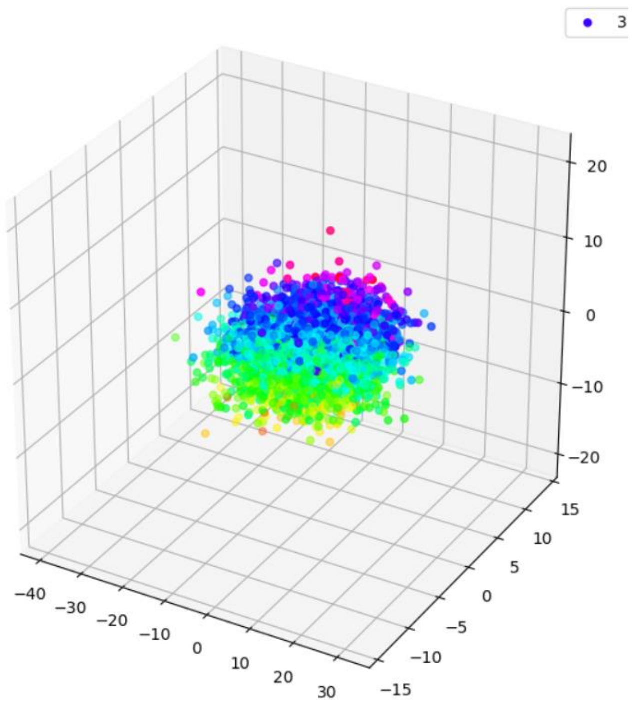
Eigen Vectors:



2D data:



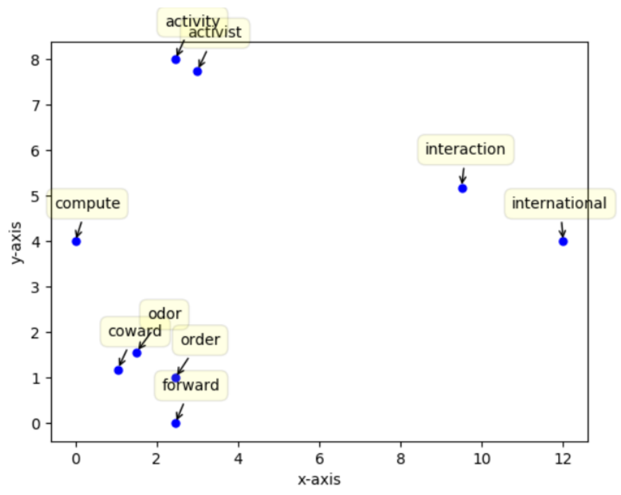
3D data:



FastMap

```
Word          Embedding
acting        [3.875   6.0625]
activist          [3.    7.75]
compute       [0. 4.]
coward        [1.04166667 1.1875     ]
forward       [2.45833333 0.         ]
interaction       [9.5    5.1875]
activity          [2.45833333 8.         ]
odor          [1.5    1.5625]
order         [2.45833333 1.         ]
international        [12.   4.]
```

Plot words on 2d plane



Optimizations and Challenges

Optimization: PCA

- Plot the original data points and new data points respectively on 3D space and 2D plane, showing the comparison between before and after execution to get the better clarity.
- Use a different data structure rather than numpy to reduce the complexity of the program

Optimization: FastMap

- We can write the farthest_distance function using recursion and update the distance in the same function. This might reduce the complexity and helps to find the solution in lesser time
- In the projection_on_hyper_plane(), the max_distance, Oa, Ob will be kept during the computation for next iteration. Which will save the computing time for farthest point selection for pivot objects.

Challenge: PCA

- Verifying the values with sklearn library was little troublesome plus we faced big challenge in Visualizing the eigen vectors on plane.
- The mainly challenge is from array operation. With the help of numpy package, it’s easy to calculate some important variable we need. But how to express and use the function is a little troublesome.

Challenge: FastMap

- Integration of farthest and embedding function together plus choosing the datastructore was the main challenge
- How to effectively get maximum distance information between objects is a big challenge.
- It’s not easy to record the distance information, projection information, and pivot objects information during the recursive step, which makes us puzzled during implementation.

PART 2: SOFTWARE FAMILIARIZATION

Library Function:

PCA

class sklearn.decomposition.**PCA**(*n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*)

Implementation:

```
from sklearn.decomposition import PCA
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt #Used for plotting graph

# intialize pca and logistic regression model
pca = PCA(n_components=2)

# fit and transform data

#load data from pca_data text file
X = np.loadtxt("pca_data.txt")

#n_componentsint, float, None or str
#Number of components to keep. if n_components is not set all components
are kept:
pca = PCA(n_components=2)
pca.fit(X)

#Transform the data from 3d to 2D
reduced_data = pca.transform(X)

# Function used for printing the 3D data
def plot3D(X):
    x_points = [each[0] for each in X]
    y_points = [each[1] for each in X]
    z_points = [each[2] for each in X]
    fig = plt.figure(figsize=(8, 8))
    ax = plt.axes(projection="3d")
    ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');
    plt.legend('3D Plot')

    plt.show()

#Function used for printing the 2D data
def plot2D(reduced_data):
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    for i in range(len(reduced_data)):
        _x = reduced_data[i][0]
        _y = reduced_data[i][1]

        plt.plot(_x, _y, 'b.', markersize=1)

    plt.show()

#Data  transformed to 2D
```

```
print("Transformed 2D array")
print(reduced_data)
#Some results to descrie the features of data
print("OUTPUT PARAMETERS")
print("Explained Variance")
print(pca.explained_variance_)
print("Singular values")
print(pca.singular_values_)
print("Get Covariance")
print(pca.get_covariance())
print("Get MEAN")
print(pca.mean_)
print("Eigen Vectors")
print(pca.components_)


plot3D(X)
plot2D(reduced_data)
```

FastMap from Github:

We didn't find a popular implementation from other libraries, but we find an implementation from Github. The implementation also followed the algorithm of one of our reference. So the idea is similar to ours.

Reference: https://github.com/mahmoudimus/pyfastmap

API Introduction

Here we list some different designs that we can absorb from them.

Dist: a NxN distance matrix, which records the distances between objects. In their code, they also use words as objects, and calculate the Levenshtein distance between words with the code: distmatrix(strings, c=lambda x, y: 1 - Levenshtein.ratio(x, y)).

Actually, we also consider to implement a NXN matrix to record down all distance between each object. It will speed up the distance information enquiry for any two objects but the disadvantage is memory consume if there are a lot of objects. Instead, we get the farthest objects in another approach which no need any iteration to get the information and also save the memory space for the object pairs with distance information

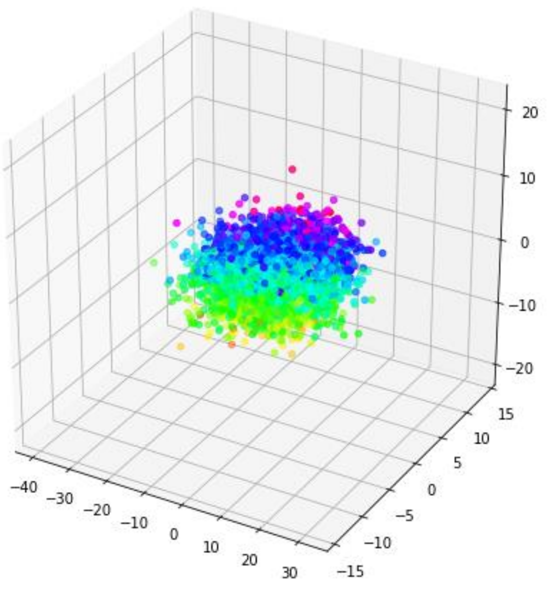
def _map(self, K): It's the main recursive operation of FastMap. In the implementation, _map mainly includes two step. The first step is recursively computing the distance based on previous projections, and the second step is project the i'th point onto the line defined by x and y. Both steps are implemented by two individual function _dist(self, x, y, k) and _x(self, i, x, y), making the process very clear.

Output:

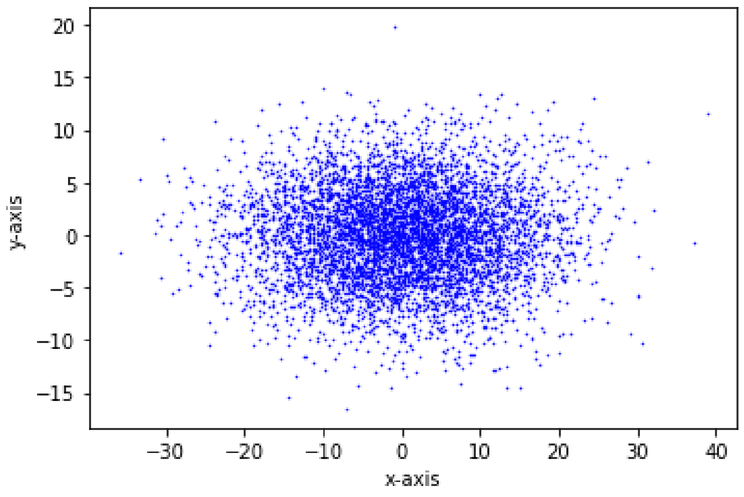
- PCA

```
Transformed 2D array
[[-10.87667009   7.37396173]
 [ 12.68609992  -4.24879151]
 [ -0.43255106   0.26700852]
 ...
 [  2.92254009   2.41914881]
 [-11.18317124   4.20349275]
 [-14.2299014    5.64409544]]
OUTPUT PARAMETERS
Explained Variance
[101.61980038  19.89921519]
Singular values
[780.77985534 345.50744124]
Get Covariance
[[ 81.24199811 -15.84081415  31.66840483]
 [-15.84081415  13.70181418 -15.26445036]
 [ 31.66840483 -15.26445036  31.36677137]]
Get MEAN
[ 0.04641608 -0.0356265   0.06334316]
Eigen Vectors
[[-0.86667137  0.23276482 -0.44124968]
 [-0.4962773  -0.4924792   0.71496368]]
```

3D Data



2D Data:



Comparison between our Implementation and sklearn pca library:

- In scikit-learn, PCA is implemented as a *transformer* object that learns n components in its fit method, and can be used on new data to project it on these components.
- Efficiency in terms of time
  - Our implementation takes **0.09185576438903809 seconds**
  - Sklean PCA Library takes **0.08406710624694824 seconds**

  As far as O/P is concerned the transformed matrix is same in both cases irrespective of the signs

PART 3: APPLICATIONS

Principal Component Analysis

- In quantitative finance, principal component analysis can be directly applied to the risk management of interest rate derivative portfolios
- A variant of principal components analysis is used in neuroscience to identify the specific properties of a stimulus that increase a neuron's probability of generating an action potential.[43] This technique is known as spike-triggered covariance analysis.

- In neuroscience, PCA is also used to discern the identity of a neuron from the shape of its action potential. Spike sorting is an important procedure because extracellular recording techniques often pick up signals from more than one neuron.
- PCA as a dimension reduction technique is particularly suited to detect coordinated activities of large neuronal ensembles. It has been used in determining collective variables, that is, order parameters, during phase transitions in the brain.

Fastmap

- Human activity recognition is one of the most popular research topics in computer vision. Automatic recognition of human actions is a very complex task due to viewpoint variations, occlusions, background interference, movement variability of the same action and ambiguity between different actions. In the paper we refer, a model relies on 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. The objective is to provide a recognition method that is both simple, fast and applicable in many scenarios.
- A near-linear time preprocessing algorithm, called FastMap can be used for producing a Euclidean embedding of a general edge-weighted undirected graph. At runtime, the Euclidean distances were used as heuristic by A* for shortest path computations.

PART 4: CONTRIBUTION

The project was planned and implemented by all group members.

1. We all discussed the design of the project.
2. The code was built in group together with discussion and peer reviews within the group.
3. Library function was studied by each member individually and collaborated to compare and analyse the difference between our results and library functions output of PCA and Fastmap algorithm.
4. Design of Fastmap and PCA class structure of input data set, dictionary mapping, calculating the farthest point method was all concluded after discussion