PCA and FastMap Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

Feb. 18, 2022

Table of Contents

1 Basic Algorithm and Code Functions Description

1.1 Implementation of PCA

1.1.1 Basic Algorithm of PCA

A. Standardize the dataset.

B. Calculate the covariance matrix for the features in the dataset.

C. Calculate the eigenvalues and eigenvectors for the covariance matrix.

D. Sort eigenvalues and their corresponding eigenvectors.

E. Pick k eigenvalues and form a matrix of eigenvectors.

F. Transform the original matrix.

1.1.2 Code Functions of PCA

- **meanNormalization(points)**

It normalizes the data points. It accepts the dataset as input. It calculates the mean value μ of all data points and assigns $(x_i - \mu)$ to x_i.

- **covarMatrix(points)**

It calculates the covariance matrix. It accepts the dataset as input.

- **firstKEigenvector(covar, k)**

It accepts a covariance matrix and the number of dimensions. First, it calculates the eigenvalues and eigenvectors for the covariance matrix. Then, it sorts eigenvalues and their corresponding eigenvectors by decreasing order. Next, it picks the first k eigenvalues and forms a matrix of eigenvectors.

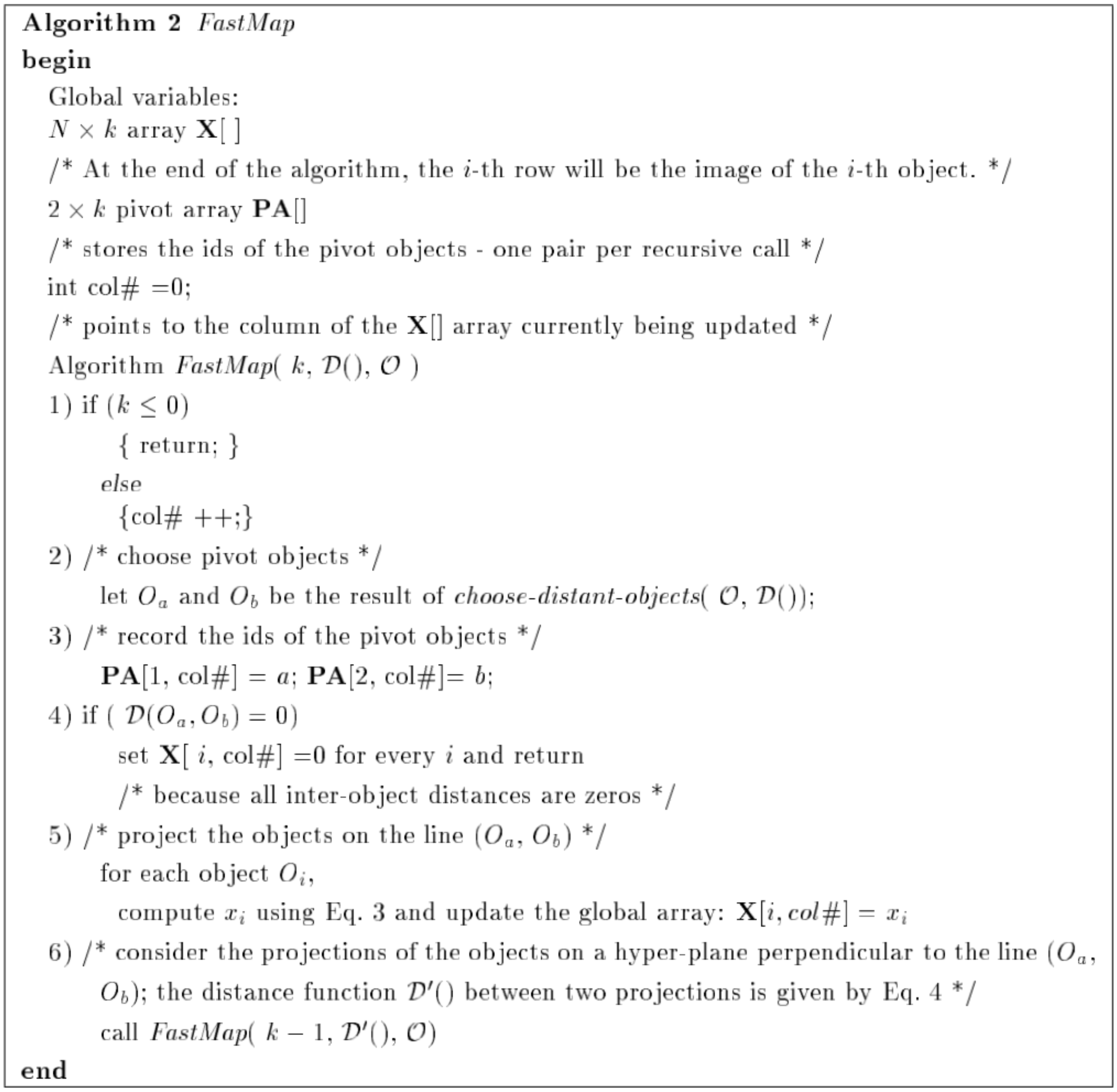
- **PCA(points, dimensions)**

It is the main function of PCA algorithm. It accepts the original dataset and the desired dimension. It calls the three function above to get the 2D projected result.

1.2 Implementation of FastMap

1.2.1 Basic Algorithm of FastMap

I implemented the fastMap program based on the original paper by Faloutsos et al. 1995.



```
Algorithm 2 FastMap
begin
  Global variables:
  $N \times k$ array $\mathbf{X}[\,]$
  /* At the end of the algorithm, the $i$-th row will be the image of the $i$-th object. */
  $2 \times k$ pivot array $\mathbf{PA}[]$
  /* stores the ids of the pivot objects - one pair per recursive call */
  int col# =0;
  /* points to the column of the $\mathbf{X}[]$ array currently being updated */
  Algorithm FastMap( $k$, $\mathcal{D}()$, $\mathcal{O}$ )
  1) if ($k \leq 0$)
         { return; }
       else
         {col# ++;}
  2) /* choose pivot objects */
       let $O_a$ and $O_b$ be the result of choose-distant-objects( $\mathcal{O}$, $\mathcal{D}()$);
  3) /* record the ids of the pivot objects */
       $\mathbf{PA}[1, \text{col\#}] = a$; $\mathbf{PA}[2, \text{col\#}] = b$;
  4) if ( $\mathcal{D}(O_a, O_b) = 0$)
         set $\mathbf{X}[\,i, \text{col\#}] = 0$ for every $i$ and return
         /* because all inter-object distances are zeros */
  5) /* project the objects on the line ($O_a$, $O_b$) */
       for each object $O_i$,
         compute $x_i$ using Eq. 3 and update the global array: $\mathbf{X}[i, col\#] = x_i$
  6) /* consider the projections of the objects on a hyper-plane perpendicular to the line ($O_a$,
       $O_b$); the distance function $\mathcal{D}'()$ between two projections is given by Eq. 4 */
       call FastMap( $k-1$, $\mathcal{D}'()$, $\mathcal{O}$)
end
```

1.2.2 Code Functions of FastMap

- **findFarthestPoints(disMatrix)**

The algorithm of this function is shown below. It accepts the distance matrix to get a pair of points that have the highest distance.

```
Algorithm 1 choose-distant-objects ( $\mathcal{O}$, dist() )
begin
  1) Chose arbitrarily an object, and declare it to be the second pivot object $O_b$
  2) set $O_a$ = (the object that is farthest apart from $O_b$) (according to the distance function
     dist())
  3) set $O_b$ = (the object that is farthest apart from $O_a$)
  4) report the objects $O_a$ and $O_b$ as the desired pair of objects.
end
```

- **projection(disMatrix, pair, point)**

It accepts the distance matrix, the line to project (consisting of two points) and an outer point. It will find the projecting coordinate based on the following formula:

$$x_i = \frac{{d_{a,i}}^2 + {d_{a,b}}^2 - {d_{b,i}}^2}{2d_{a,b}}$$

- **updateDisMatrix(disMatrix, X)**

It accepts the distance matrix and the projecting coordinate gotten from projection(). This function updates the distance matrix by the following formula after each dimension:

$$(\mathcal{D}'(O_i', O_j'))^2 = (\mathcal{D}(O_i, O_j))^2 - (x_i - x_j)^2 \quad i, j = 1, \ldots, N$$

- **fastMap(disMatrix, dimension)**

It is the main function of fastMap algorithm. It accepts the original distance matrix and the desired dimension. It calls the three functions above to get the updated distance matrix and the 2D result.

2 Data Structure Used

2.1 PCA data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[point#0 x,y,z], [point#1 x,y,z], …, [point#N-1 x,y,z]]

B. The covariance matrix is stored in the data structure of n*n NPArray.

C. The original eigenvector and sorted eigenvector is stored in the data structure of n*n NPArray. The format is:

[(eigenvector 1), (eigenvector 2), …, (eigenvector n)]

D. The first-k-sorted eigenvector is stored in the data structure of NPArray. The format is:

[(eigenvector 1), (eigenvector 2), …, (eigenvector k)]

E. The eigenvalue is stored in the data structure of NPArray. The format is:

[eigenvalue 1, eigenvalue 2, …, eigenvalue n]

2.2 FastMap data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[pointA, pointB, distance], [pointA, pointC, diatance], …, [pointN-1, pointN, distance]]

B. The distance matrix is stored in the data structure of 2D NPArray. The format is:

[(O_1, O_2), (O_1, O_3), … ,(O_i, O_j)]

3 Challenges Faced and Optimizations

3.1 Challenges

A. The PCA algorithm involves a lot of matrix operations, and I can choose to implement it myself or use NumPy's built-in functions. But many times I need to consult the web documentation to know what functions are available in NumPy.

B. How to evaluate the result of fastMap algorithm is difficult because there is no package implemented in this algorithm.

3.2 Optimizations

A. The NPArray is faster than List in access and computation.

B. In the fastMap program, I keep the value of O_i, O_j during the computation for the next iteration. It will save time for computing.

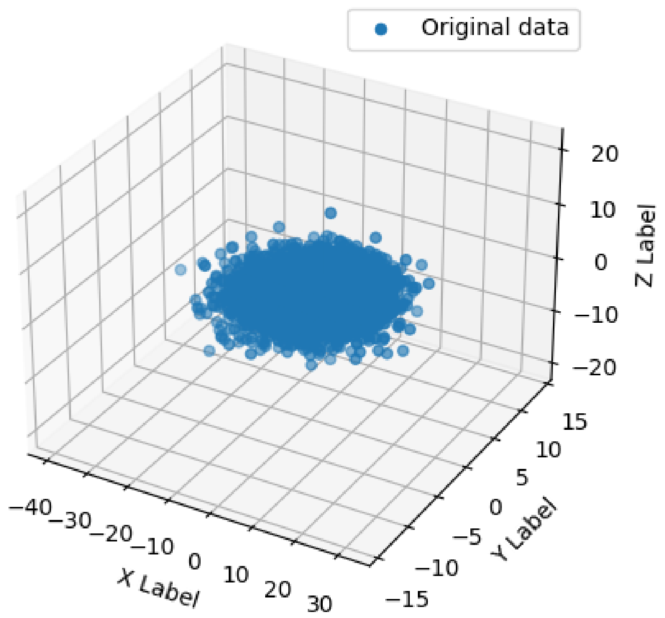
4 Execution of my algorithm and results

4.1 Execution of PCA program

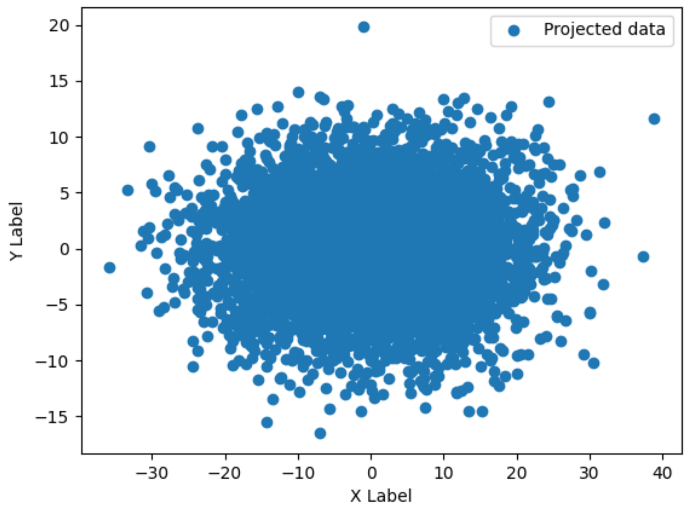
- Print of result:

```
Sorted eigenvector #1=  [-0.86667137  0.23276482 -0.44124968]
Sorted eigenvector #2=  [-0.4962773  -0.4924792   0.71496368]
results=
 [[-10.87667009   7.37396173]
 [ 12.68609992  -4.24879151]
 [ -0.43255106   0.26700852]
 ...
 [  2.92254009   2.41914881]
 [-11.18317124   4.20349275]
 [-14.2299014    5.64409544]]
```

- Image of original 3D data:



- Image of projected 2D data:



4.2 Execution of fastMap program

- Print of result:

```
2D result:
[[3.875, 6.0625],
 [3.0, 7.749999999999999],
 [0, 4.0],
 [1.0416666666666667, 1.1875],
 [2.4583333333333335, 0],
 [9.5, 5.1875],
 [2.4583333333333335, 8.0],
 [1.5, 1.5624999999999996],
 [2.4583333333333335, 1.0],
 [12.0, 4.0]]

Word Mapping:
"acting":[3.875, 6.0625]
"activist":[3.0, 7.749999999999999]
"compute":[0, 4.0]
```

"coward":[1.0416666666666667, 1.1875]

"forward":[2.4583333333333335, 0]

"interaction":[9.5, 5.1875]

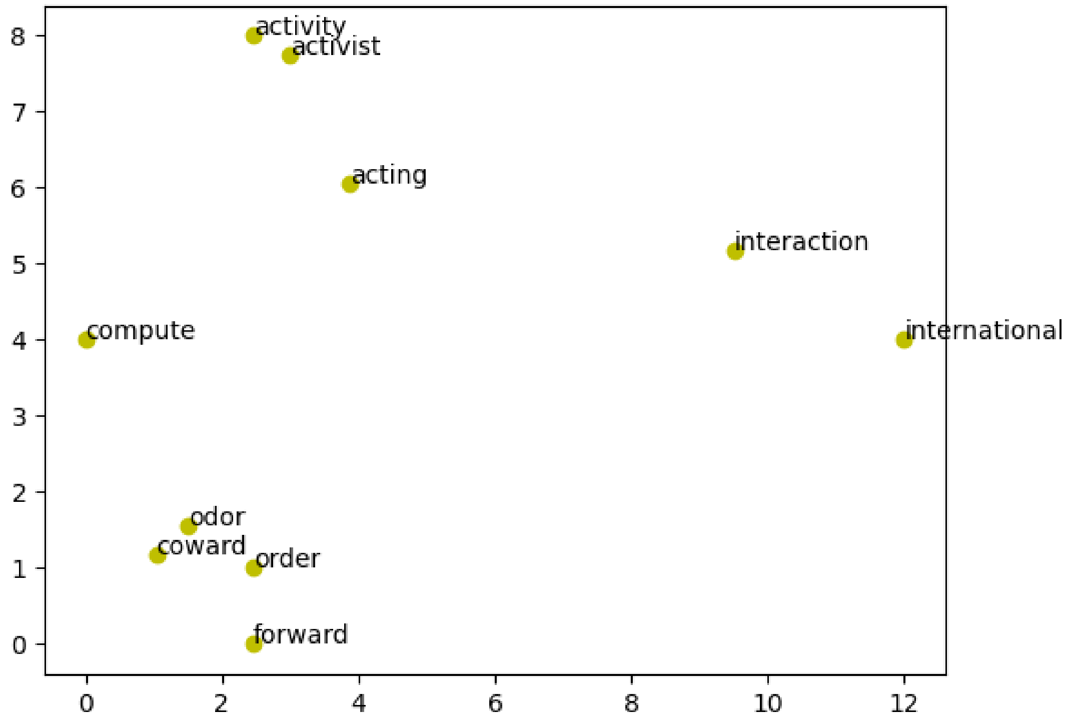
"activity":[2.4583333333333335, 8.0]

"odor":[1.5, 1.5624999999999996]

"order":[2.4583333333333335, 1.0]

"international":[12.0, 4.0]

- Image of word mapping:



5 Comparison between my algorithm and scikit-learn

5.1 PCA Comparison

- The code of scikit-learn in PCA is:

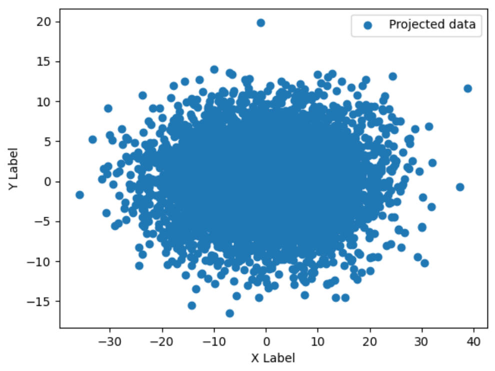
```
from numpy import genfromtxt
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
dimension = 2
if __name__ == '__main__':
    X = genfromtxt('pca-data.txt', delimiter='\t')
    pca = PCA(n_components=dimension)
    pca.fit(X)
    for idx, pc in enumerate(pca.components_):
        print('Vector {0}: {1}'.format(idx + 1, pc))
    transformed = pca.transform(X)
    print("result:\n", transformed)
    fig, ax = plt.subplots()
    plt.xlabel('X Label')
    plt.ylabel('Y Label')
    projection = ax.scatter(transformed[:, 0], transformed[:, 1], c='g')
    ax.legend([projection], ['Projected data'])
    plt.savefig('2dData_sklearn.png')
    plt.close()
```

- The result is:

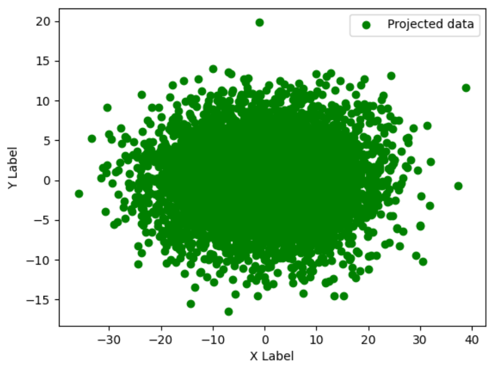
```
Vector 1: [-0.86667137  0.23276482 -0.44124968]
Vector 2: [-0.4962773  -0.4924792   0.71496368]
result:
 [[-10.87667009   7.37396173]
 [ 12.68609992  -4.24879151]
 [ -0.43255106   0.26700852]
 ...
 [  2.92254009   2.41914881]
 [-11.18317124   4.20349275]
 [-14.2299014    5.64409544]]
```

The results are same as mine.

- The comparison of image between my program and scikit-learn is:



Mine



scikit-learn

5.2 FastMap Comparison

I did not find any Python package or library that implements the fastMap algorithm. Actually, there are really few materials about fastMap. Therefore, I did not make any comparison.

6 Application of PCA and fastMap

6.1 Application of PCA

- Spike-triggered covariance analysis in Neuroscience. It uses a variant of PCA in Neuroscience to identify the specific properties of a stimulus that increase a neuron's probability of generating an action potential.
- Quantitative Finance. The application of PCA in finance includes: Analyzing the shape of the yield curve, Hedging fixed income portfolios, Implementation of interest rate models, Forecasting portfolio returns, Developing asset allocation algorithms, and Developing long-short equity trading algorithms.
- Image Compression. PCA can be applied to compress the pixels of an image to reduce the size of the image file while maintaining the basic information of the image.

6.2 Application of fastMap

According to what the professor told in class, fastMap can be applied to:

- Speeding up shortest-path computations in goal-directed A* search.
- Solving useful graph-theoretic combinatorial problems using geometric interpretations.
- Efficiently computing measures of centrality on large graphs.
- Embedding directed graphs in potential fields.
- New algorithms for community detection and block modeling.
- New ways of doing machine learning on graphs.

7 Contributions

All works were completed by Chaoyu Li.