Technical Specification and Study Report

[PCA & FastMap]

Author: [Cheng-Lin Li (9799716705), Jianfa Lin (6950769029)]
Date: [Oct., 11, 2016]
Version: [1.0]

Document Control

Author

Position	Name	USC ID
Student	Cheng-Lin Li	9799716705
Student	Jianfa Lin	6950769029

Revision history

Version	Issue date	Author/editor	Description/Summary of changes
0.8	2016/10/8	Jianfa Lin	Finish sections of introduction and functionality
0.9	2016/10/9	Jianfa Lin	Include the implementation of PCA and FastMap.
0.91	2016/10/10	Jianfa Lin	Include the software familiarization
1.0	2016/10/11	Jianfa Lin	First release

Related documents

Document	Description
PCA.py	PCA implementation by Python 3.5
FastMap.py	FastMap implementation by Python 3.5
pca-data.txt	Data Set for PCA
fastmap-data.txt	Data Set for FastMap
fastmap-wordlist.txt	Data Set for FastMap

Contribution

Name	USC ID	Labour
Cheng-Lin Li	9799716705	Implement the code
Jianfa Lin	6950769029	Write the document

Table of Contents

1 INTRODUCTION

1.1 Objectives

Implement the PCA algorithm and FastMap algorithm for reducing the dimensionality of data and giving the representation of data in a fewer-dimension space.

a. 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.
b. 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.

The data set for PCA is a 3D points set, in which the data's coordinates on three dimensions are given. The data set for FastMap indicates the distance between every two objects. The first two columns in each line of the data file represent the IDs of the two objects; and the third column indicates the symmetric distance between them.

2 Functionality

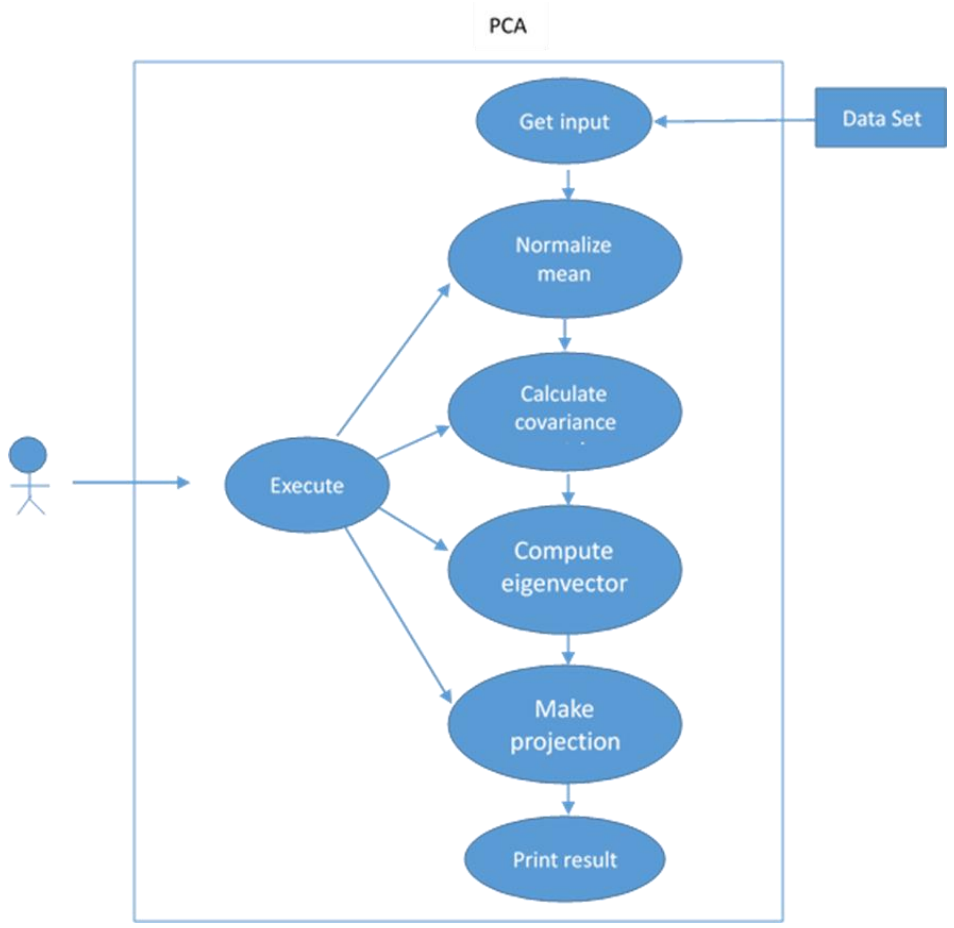
2.1 Description: PCA

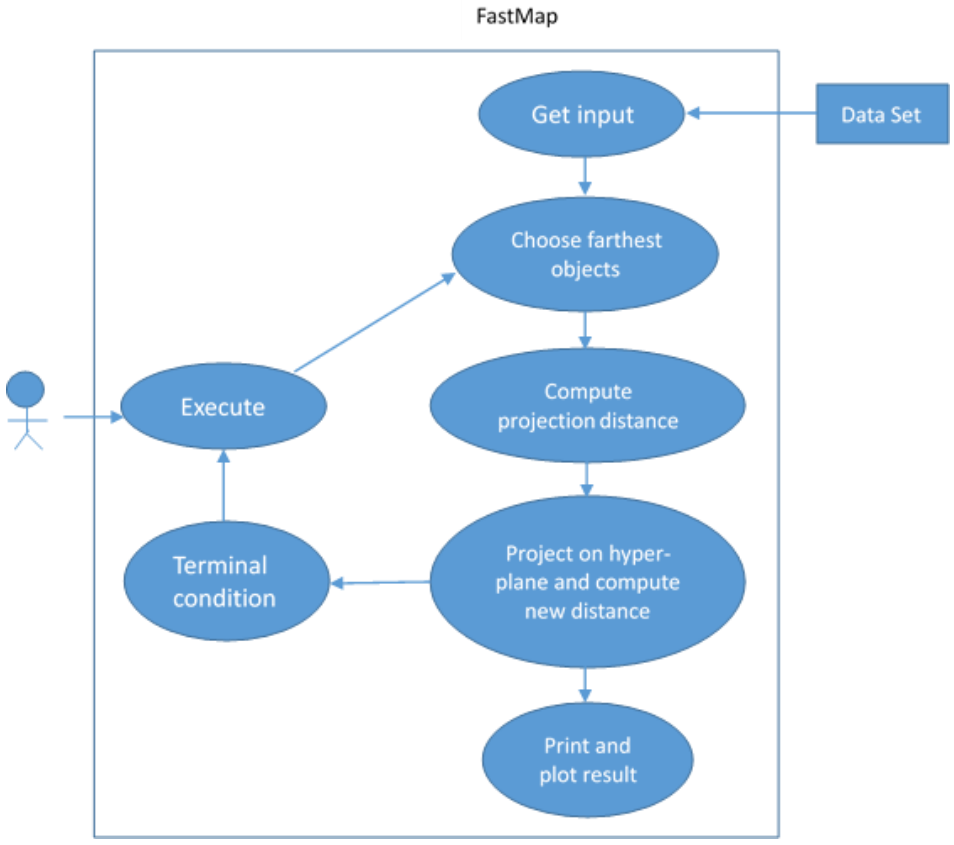
1. Assign k as the number of dimension that want to reduce to.
2. Get all data points from input.
3. Normalize data point: calculate the mean value μ of all data points and assign $(X^{(i)} – μ)$ to $X^{(i)}$. ($X^{(i)}$ stands for point i)
4. Compute covariance matrix Σ of all data points.
5. Compute eigenvalue & eigenvectors of matrix Σ, which is denoted as S and U.
6. Take first k column S and corresponding of U matrix as U_{reduce}.
7. Calculate the k-dimension representation of each point $X^{(i)}$ with $Z^{(i)} = U_{reduce} * X^{(i)}$.
8. Print and plot the result

2.2 Description: FastMap [2]

1. Assign k as the number of dimension that want to reduce to.
2. Get all data points from input.
3. Find 2 objects O_a & O_b are farthest apart from each other as pivot objects.
4. Project the object O_i on line (O_a, O_b) and calculate the distance from O_i to O_a.
5. Consider a (n-1)-dimension hyperplane H that is perpendicular to the line (O_a, O_b) and then project objects on the plane.
6. Calculate the distance between each points on the new hyperplane H.
7. Recursively apply step 3 ~ 6 (k) times for k dimensions.
8. Print and plot the result.

2.3 Use Case

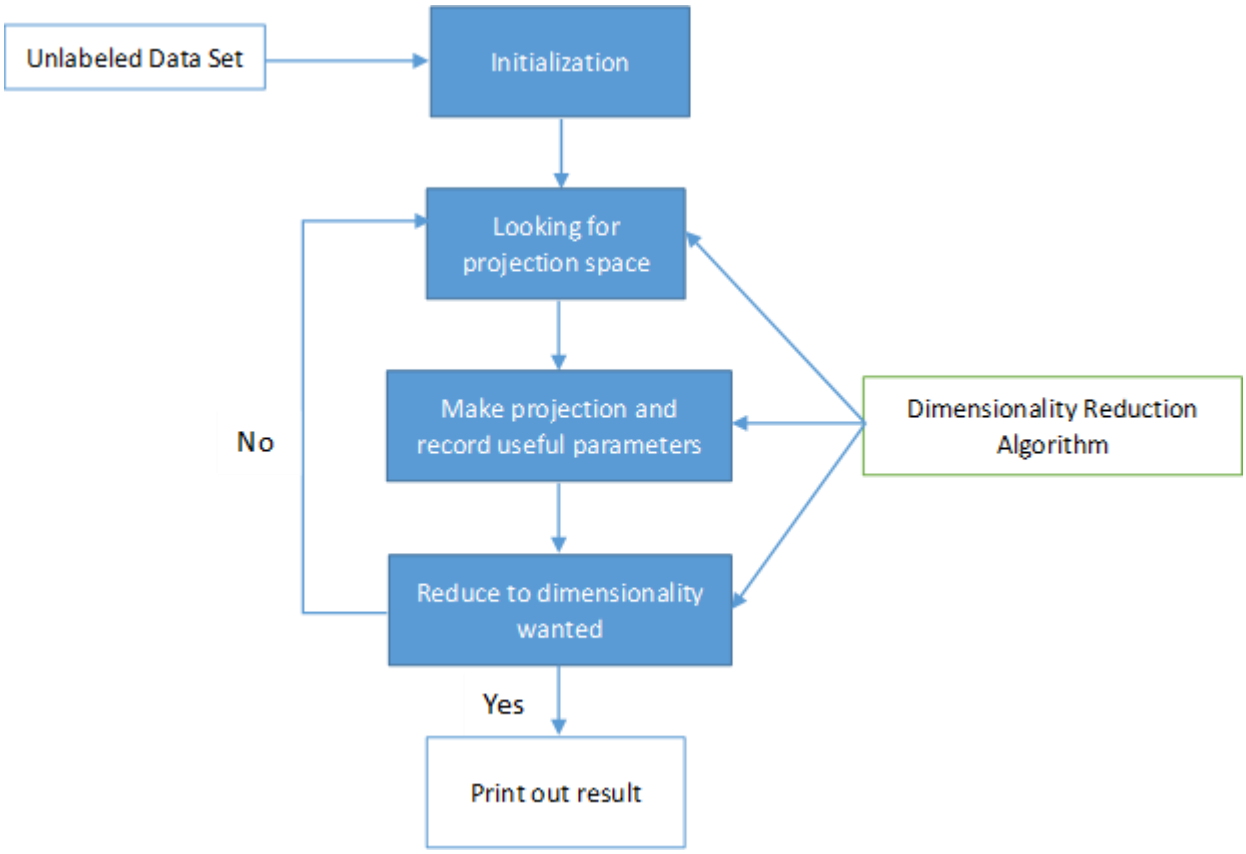

PCA
Get input
Data Set
Normalize mean
Calculate covariance
Execute
Compute eigenvector
Make projection
Print result


FastMap
Get input
Data Set
Choose farthest objects
Execute
Compute projection distance
Terminal condition
Project on hyper-plane and compute new distance
Print and plot result

3 Architecture

3.1 Architecture Overview

Here is the generalized architecture diagram for dimensionality reduction algorithms.



4 Design

4.1 Data Structure

4.1.1 PCA

1. All data points store in an array.
    a. The data structure is [point 0, point 1, …, point N-1]
    b. For each point, the data structure is [x, y, z], in which x, y and z stand for the three coordinates respectively of points
    c. mn_x is the data set after mean normalization.
2. Covariance stores in an array.
    a. covar: Store covariance of all data points, which is a n*n matrix.
3. Eigenvactor stores in an array.
    a. Store covariance of all data points, which is a n*n matrix.
    b. eigenvector = [(eigenvector 1), (eigenvector 2), …, (eigenvector n)]. n is the dimension of data points.
    c. sorted_eigenvector = [(eigenvector 1), (eigenvector 2), …, (eigenvector n)]
    d. sorted_k_eigenvector = [(eigenvector 1), (eigenvector 2), …, (eigenvector k)]. k is the dimension we want to reduce to.
4. Eigenvalue stores in an array.
    a. Store weighted values of different eigenvector. covariance of all data points, which is a n*n matrix.

4.1.2 FastMap

1. All object pairs store in an array.
    a. o_pair = [pair 1, pair 2, …, pair n]
    b. The data structure of each pair is [x, y]. x, and y stand for two object IDs.
2. Distances store in an array.
    a. Store distance of each object pair.
    b. dist = [distance of pair 1, distance of pair 2, …, distance of pair n].
    c. obj_k_d is an array stores object distances scale to k dimensions.
3. New distance information stores in an array.
    a. _new_dist_set = [distance between pair 1 of projections in new hyper-plane, …, distance between pair n of projections in new hyper-plane]
4. Objects store in a set.
    a. Store the label of total objects.
    b. obj_set = [object 1, …, object k]

4.2 Procedures

4.2.1 PCA [2]

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

4.2.2 FastMap

1. Get all data points from input file, fastmap-data.txt, into system. The data file contains 45 pairs of objects and distances between each pair.
2. Find the farthest distance between two objects O_a and O_b.
3. Project objects into points on the line that passes through pivot objects O_a and O_b, and calculate the new distance x_i between projections and O_a.
4. Project all data points to a hyper-plane H that is perpendicular to the line (O_a, O_b), and calculate the distances between projections on H. $D'(O_i', O_j')^2 = D(O_i, O_j)^2 – (x_i – x_j)^2$. Record the distances in a set: _new_dist_set.
5. Termination:
   a. Condition:
      i. If the value of column of x array with k dimension hasn't reach k
      ii. If _max_dist equals to 0 on new hyper plane.
   b. Else repeat step 2 ~ 4.
6. Print the distance between objects in 2D plane.
7. As the optional function, get objects from file, fastmap-wordlist.txt, plot the words onto a 2D plane.

4.3 Optimizations and Challenges

4.3.1 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.

4.3.2 Optimization: FastMap

- Except first dimension the program selects the farthest objects as pivot objects from existing distance information base on reference paper. For rest of iterations, we get the maximum distance and pivot objects directly from self.max_dist, self.Oa, self.Ob that were computed during the "projection_on_hyper_plane()" step.
- 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.

4.3.3 Challenge: PCA

- 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.

4.3.4 Challenge: FastMap

- 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.

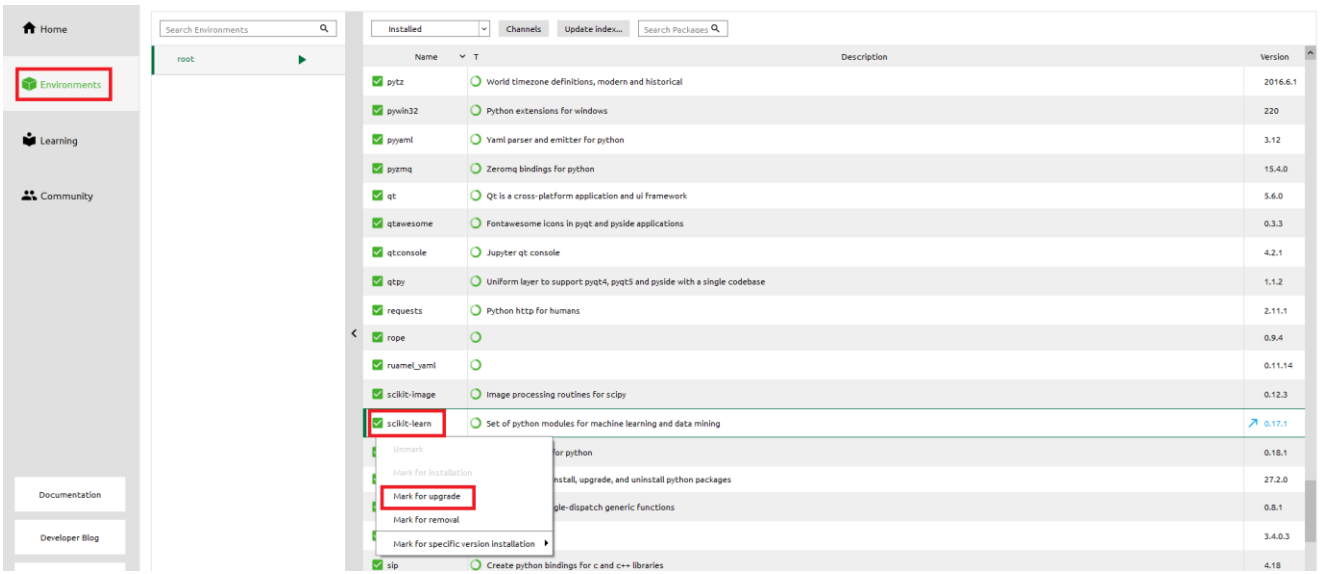
5 Usage Instruction

The implementation and clustering test code will leverage Python 3.5 with a lot of libraries. We recommend the latest “Anaconda version 4.2.0” data science platform with Python version 3.5 as execution environment. It is a library management environment and collects 175 libraries for developer to build up their own code. Spyder 3.0.0 includes in the platform as IDE.

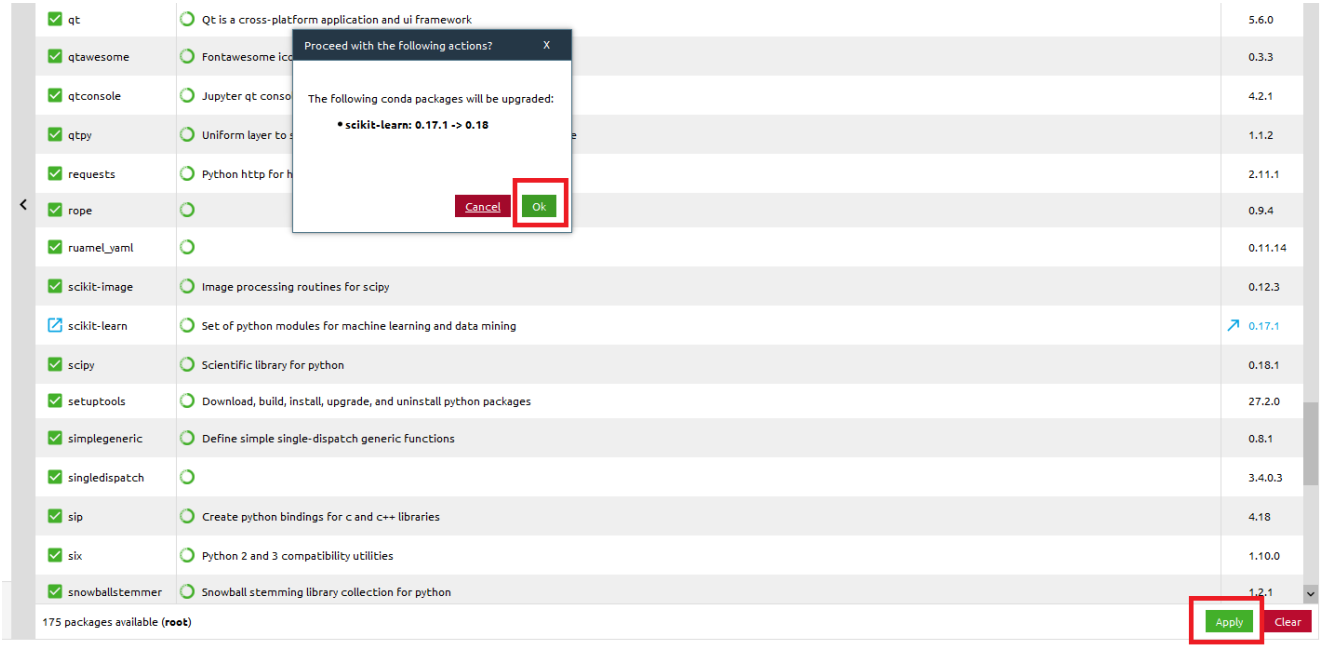
The URL of this package is https://www.continuum.io/downloads

5.1 Upgrade scikit-learn to version 0.18

Due to the latest official release of scikit-learn version 0.18 which does not include in the latest Anaconda version 4.2.0. Please upgrade scikit-learn in Anaconda environment.



Apply the upgrade.

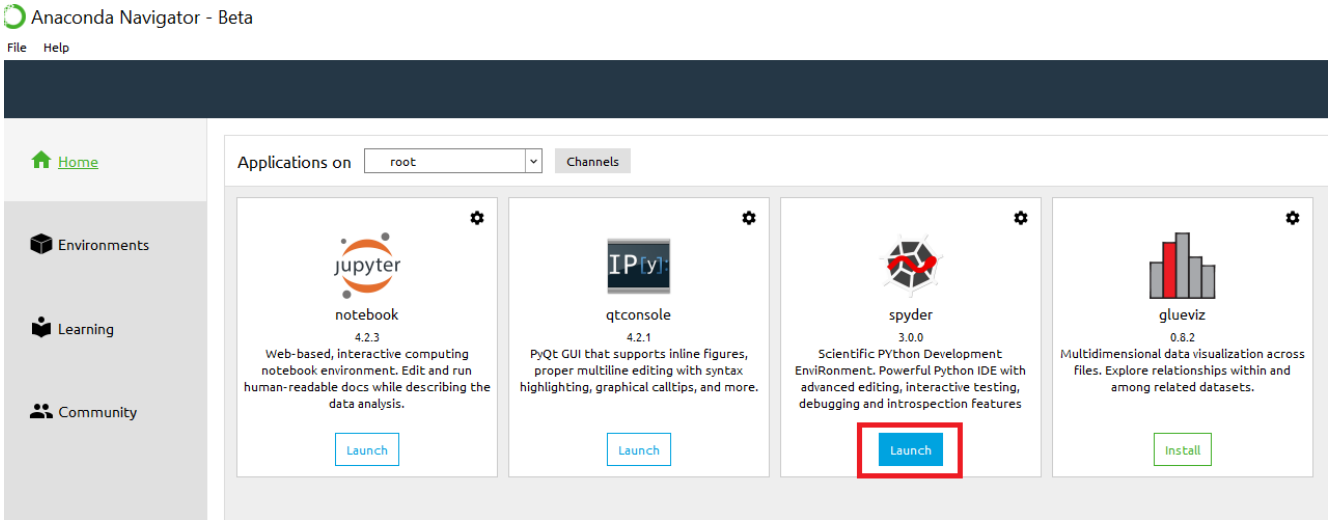


5.2 Script Files and Dataset File in same folder

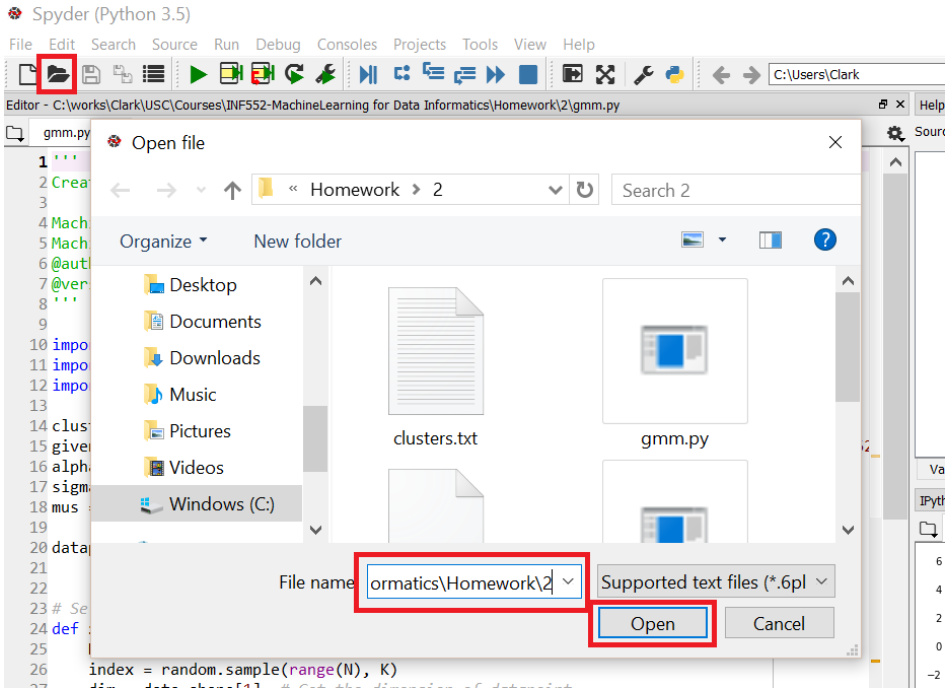
- PCA.py (Implementation code)
- pca-data.txt
- FastMap.py (Implementation code)
- fastmap-data.txt
- fastmap-wordlist.txt

5.3 Execution from Spyder version 3.0

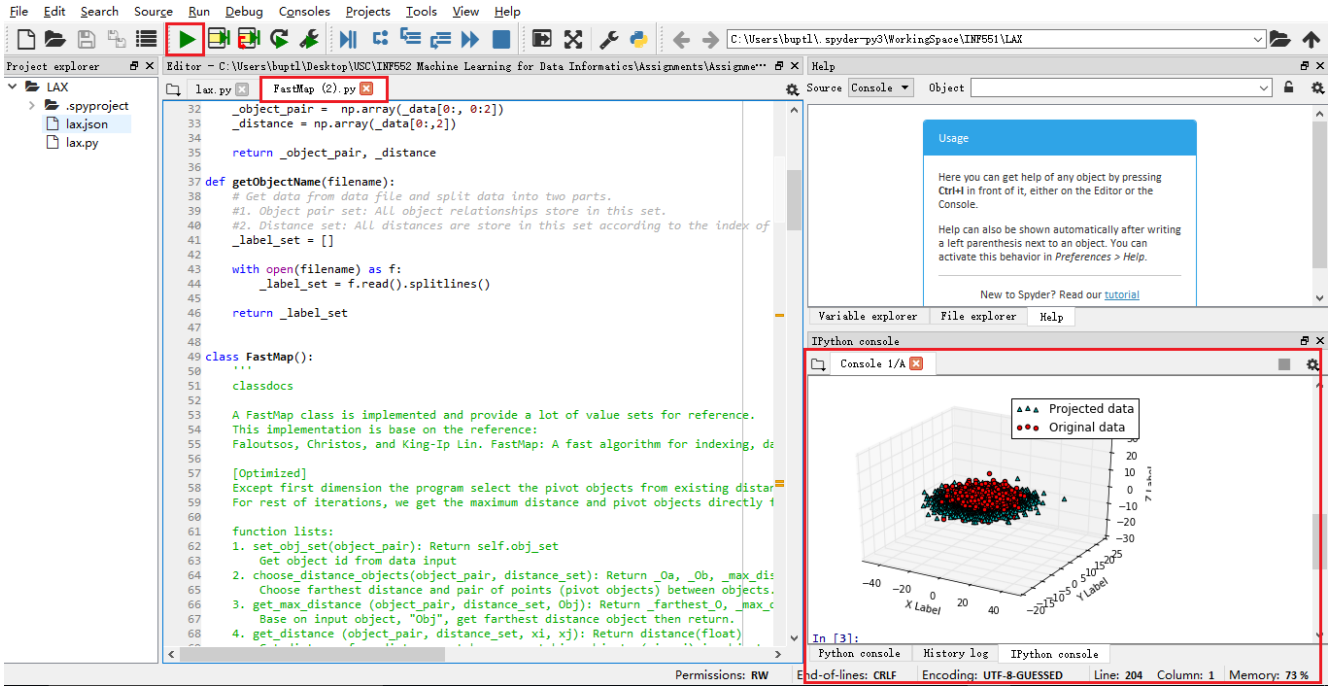
1. Under the “Home” of Anaconda Navigator, Click “Launch” to enable spyder 3.0.0 environment.



2. In Spyder IDE, please select open file icon on tool bar and open implementation and testing files.



3. Please select the implementation code you want to execute, then click the “Run file” icon. The result will show in IPython console window.



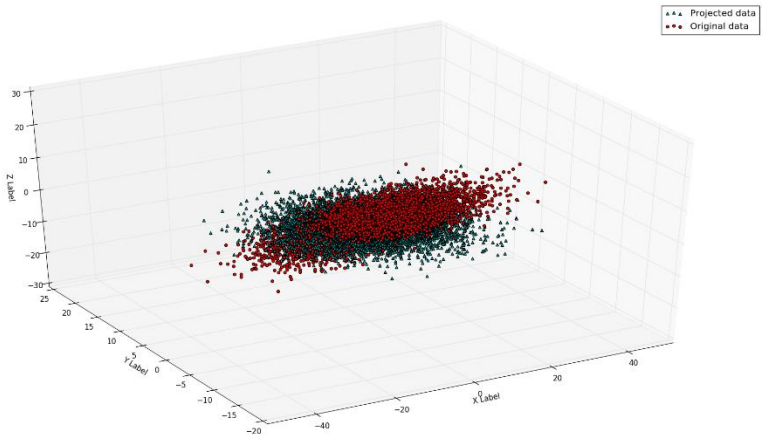
5.4 Execution Results

5.4.1 PCA

Here is the 2D representation of data points we can get and the plot of these points.

Dimensions reduce to (2), and results as below:

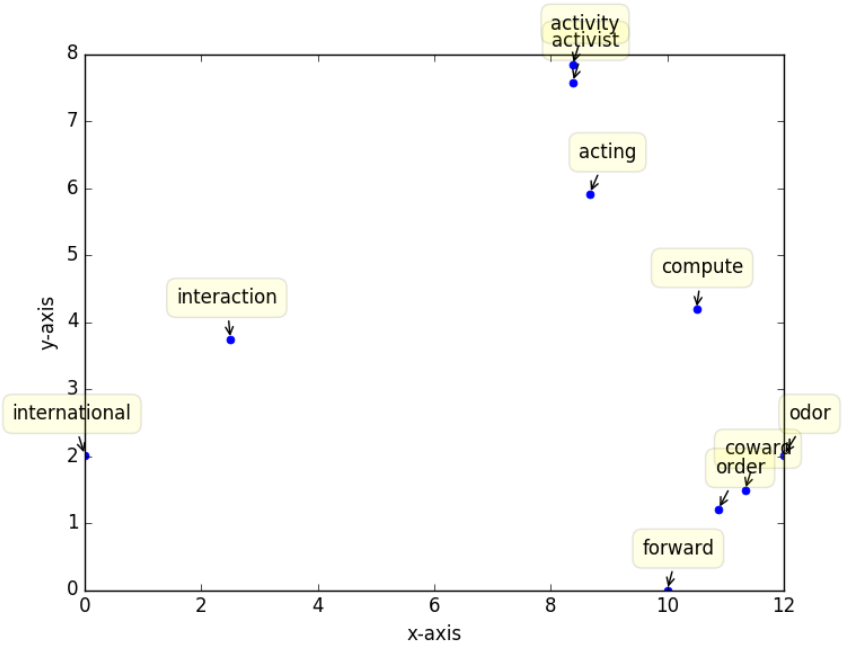
```
[[-10.87667009   7.37396173]
 [ 12.68609992  -4.24879151]
 [ -0.43255106   0.26700852]
 ...,
 [  2.92254009   2.41914881]
 [-11.18317124   4.20349275]
 [-14.2299014    5.64409544]]
```



5.4.2 FastMap

Here is the 2D coordinates of data points we get and the plot of words on 2D plane.

```
The  2  dimensions result =
 [[  8.66666667   5.91497729]
 [  8.375        7.57789976]
 [ 10.5          4.18888905]
 [ 11.33333333   1.48938277]
 [ 10.           0.        ]
 [  2.5          3.74207421]
 [  8.375        7.83322252]
 [ 12.           2.01066674]
 [ 10.875        1.20280957]
 [  0.           2.01066674]]
```



6 Software Familiarization

6.1 PCA in scikit-learn library [3]

6.1.1 Overview

We find a good implementation from scikit-learn parckage, which is based on Numpy library, scipy library. We take it as a comparison to our implementations.

6.1.2 API introduction

There are several interesting parameters can be assigned into this implementation.

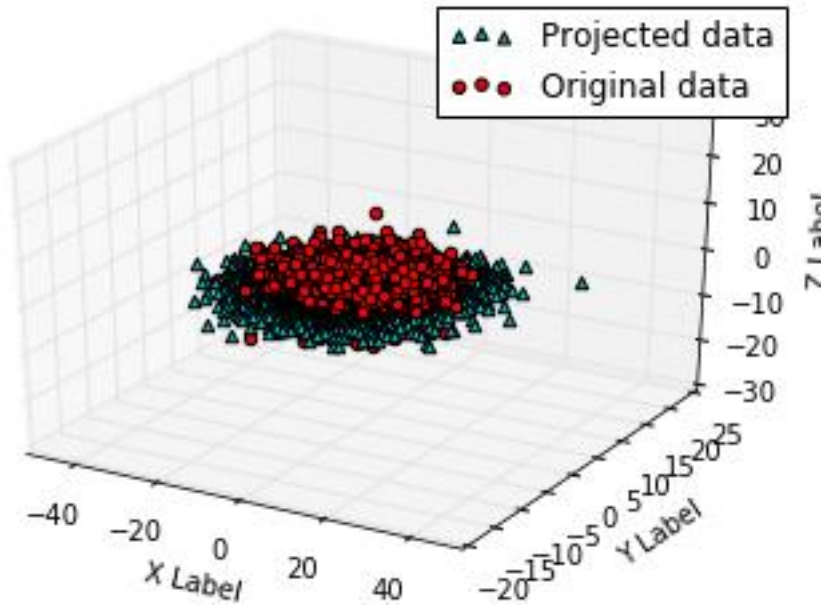
1. **n_components**: Number of components to keep.
2. **whiten**: When True (False by default) the `components_` vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.maximum iteration number can be restricted. Default is 300.
3. **svd_solver**: {'auto', 'full', 'arpack', 'randomized'}, default is ‘auto’. the solver is selected by a default policy based on `X.shape` and `n_components`: if the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension of the data, then then more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards..

6.1.3 Execution results

```
data = np.genfromtxt('pca-data.txt', delimiter='\t')
pca = PCA(n_components=2)
pca.fit(data)
projection = pca.transform(data)
print('Results of implementation from sklearn as below:\n', projection)
```

The result is the same as the results of our implementation:

Results of implementation from sklearn as below:
[[-10.87667009 7.37396173]
[12.68609992 -4.24879151]
[-0.43255106 0.26700852]
...,
[2.92254009 2.41914881]
[-11.18317124 4.20349275]
[-14.2299014   5.64409544]]



6.1.4 Advantages

- It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract. If the input data is larger than 500x500, and the number of components to extract is lower than 80% of the smallest dimension of the data, then then more efficient 'randomized' method is enabled.
- Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

6.2 FastMap from Github [4]

6.2.1 Overview

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.

6.2.2 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.

7 Application

7.1 Applications of PCA in GIS [5]

When Principal Component Analysis used in conjunction with GIS modeling, the entire approach produces hierarchies of the administrative units, which by mapping allow for the identification of hotspots (e.g., underdeveloped regions etc.) that are at the core of intervention policies. Since PCA results into the determination of the most influential variables and their weights based on the percentage of variability explained, the outputs are then embedded in a Geographical Information System model to produce and map a hierarchy of spatial subunits based on the values of the most influential variables for each subunit.

The paper we refer attempted to introduce a methodology based on using PCA in conjunction with GIS modeling to assess the level of development within the territorial subunits of a given region with different sizes, testing the hypothesis according to which the level of development cannot be accurately described from a unique standpoint economic, social, cultural etc.

7.2 Applications of FastMap in human action recognition [6]

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.

8 Reference

[1] Alpaydin E. Introduction to machine learning[M]. MIT press, 2014.

[2] Faloutsos C, Lin K I. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets[M]. ACM, 1995.

[3] scikit-learn, version 0.18, PCA, URL http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[4] A python implementation of FastMap on Github, URL https://github.com/mahmoudimus/pyfastmap

[5] Petrişor A I, Ianoş I, Iurea D, et al. Applications of principal component analysis integrated with GIS[J]. Procedia Environmental Sciences, 2012, 14: 247-256.

[6] Belhadj L C, Mignotte M. Spatio-temporal fastmap-based mapping for human action recognition[C]//Image Processing (ICIP), 2016 IEEE International Conference on. IEEE, 2016: 3046-3050.