

ASSIGNMENT 3: DIMENSION REDUCTION

PART 1: IMPLEMENTATION

All codes are written in Python 3.

In this assignment, we produced 2 version of implementation for each dimension reduction method. In Principle Component Analysis, both of our codes have nice time complexity and therefore the running times are barely the same with the one of scikit-learn package. On the other hand, not only the result of Fastmap might vary between implementation (which is reasonable since Fastmap has a random starter element in it), but also the running times are considerably different.

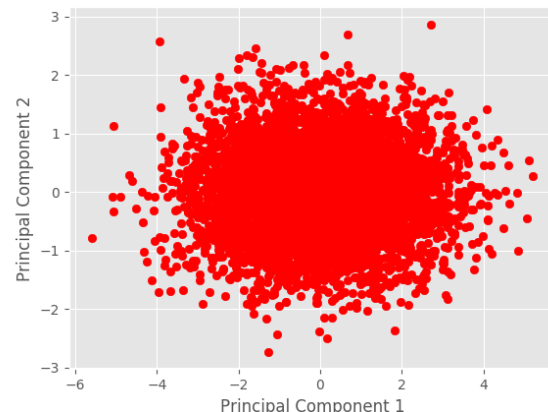
DIRECTION OF THE FIRST TWO PRICIPLE COMPONENTS

[0.53627113 -0.79599505], [-0.57568437 -0.58816792]

PCA #1 (PCA_FASTMAP.PY)

THE DATA STRUCTURES USED AND DESCRIPTION

I use pandas.DataFrame as the data structure. Based on the formula, I firstly calculate the eigenvalues and its corresponding eigenvectors. Secondly, sort the eigenvalues and select first two corresponding eigenvectors with bigger eigenvalues.

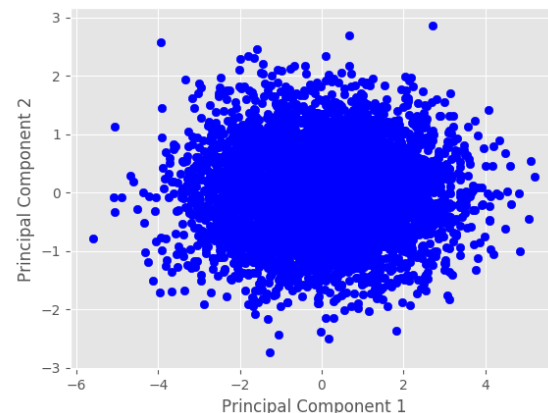


PCA #2 (PCA1.PY)

THE DATA STRUCTURES USED AND DESCRIPTION

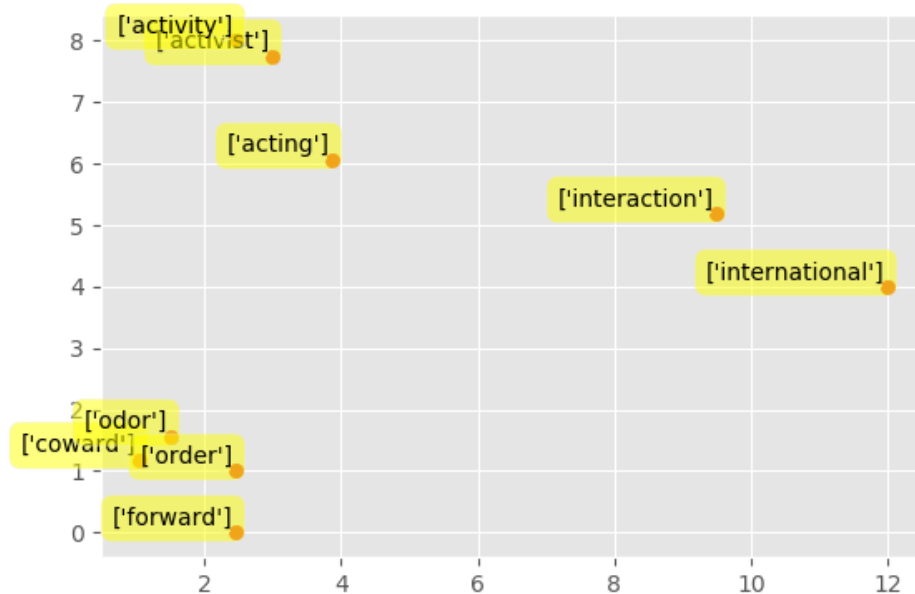
I stored the input data as an object of pandas.DataFrame, and do the calculations with numpy.

The dimension reduction procedure has no big difference with the previous one. However, there is one thing I would like to emphasize, that is, the importance of standardization.



Since PCA is trying to capture the total variance in the set of variables, it requires that the input variables have similar scales of measurement. In other words, variables whose numbers are larger will have much bigger variance just because the numbers are so big. Otherwise the biggest scale would overwhelm the PCA.

FASTMAP #1 (PCA_FASTMAP.PY)



DESCRIPTION

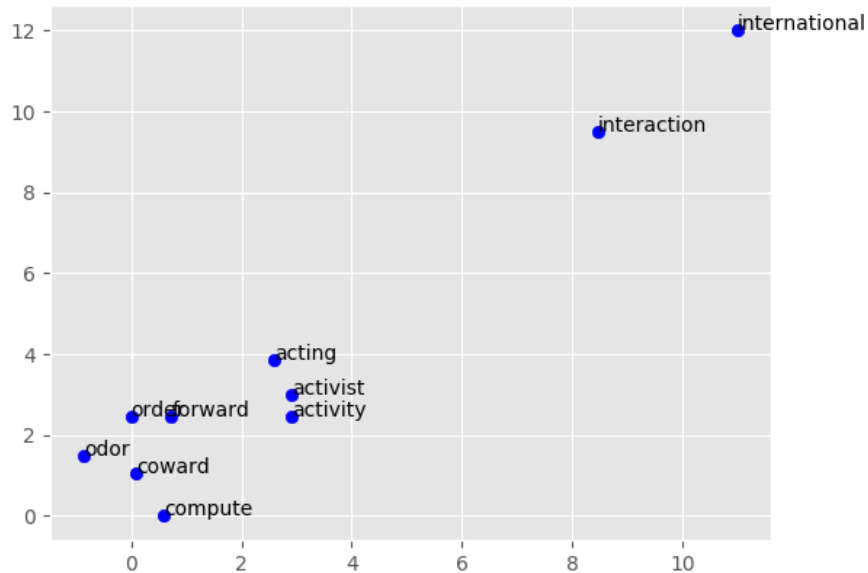
Step 1: Identify the farthest pair.

Step 2: Calculate the new distance between every two data points where the farthest pair is the coordinate.

Recursive these steps twice in order to get x-coordinate and y- coordinate of each data point.

CHALLENGE:

At the beginning, I was confused about the meaning of x-axis and y-axis, and the number of iterations required to map words into a 2D plot. Afterward, I realized that the 2D plot simply shows the distance between each two words. More specifically, in the graph, the first dimension would be the line between the two most distant points, and all the other points should be projected to the according dimension. As a consequence, I need to loop the algorithm twice. The 1st loop computes the X-axis and the 2nd loop computes the Y-axis.



THE DATA STRUCTURES USED AND DESCRIPTION

DataFrame of the pandas package is used to store the distance between words, and List stores the word list. Step 0: Arbitrarily pick up a point to start.

Step 1: Find the furthest pair of points heuristically.

Step 2: Projection. Take the line between the pair as the coordinate, and project all the points on to it by the following formula:

$$X_i = d_{ai} \cos \theta = \frac{d_{ai}^2 + d_{ab}^2 - d_{ib}^2}{2d_{ab}}, a \text{ stands for the point of coordinate zero}$$

Recursive the two step twice.

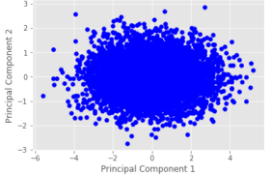
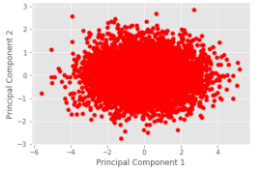
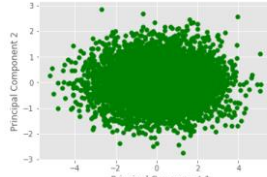
CHALLENGE

The biggest challenge for me is indexing and selection of data. Since the input distance data has two columns representing word IDs, it took me a lot of time figuring out how to efficiently iterate through the data and extract the corresponding distance. In the end, I am confident to say that my code is nice because its running time is even faster than the package I found!


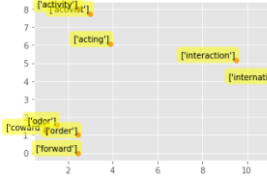
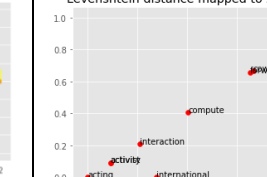
Another challenge I faced is that I was confused by a sentence in the homework description, which says, *“When computing the coordinates, please always use the object with the smaller ID of the pair as the point of coordinate zero.”* This is due to my understanding of the fact that the first point should be chosen randomly. If the second point is bigger and shows up twice, how can the first point still locates in (0, 0) afterward? Soon I found out this is not even a problem. The sentence is just a hint for us to project points.

PART 2: COMPARISON

PCA

	PCA1.py	PCA_FastMap.py	sklearn.decomposition.PCA
Operation Time(sec)	0.063sec	0.059sec	0.052sec
Output Result			
Eigenvectors	Dimension1: [0.53627113 -0.79599505] Dimension2: [-0.57568437 -0.58816792]		Dimension1: [-0.53627113 -0.79599505] Dimension2: [0.57568437 -0.58816792]
Conclusion	The outputs are almost the same. Even though our outputs' value are exactly the same with that of scikit-learn package, the plus or minus sign differs. To be more specific, see the following example: Our result: Dimension1: [0.53627113 -0.79599505] SklearnPCA: Dimension1: [-0.53627113 -0.79599505]		

FASTMAP

	FastMap1.py	PCA_FastMap.py	FastMap.py (package)
Operation Time(sec)	0.06sec	0.43sec	0.10sec
Output Result			
Big O Notation	$O(kN)$	$O(kN!)$	$O(kN)$
Conclusion	Outputs change	Outputs remain the same	Outputs change

Note that N = number of data point, k = dimension.

PART 3: APPLICATION

PRINCIPAL COMPONENT ANALYSIS(PCA):

PCA provides a significant way to eliminate data dimensions. PCA is one of the most successful techniques that have been used in image recognition and compression. The purpose of PCA is to reduce the large dimensionality of the data space (observed variables) to the smaller intrinsic dimensionality of feature space (independent variables). The functions that PCA can do are prediction, redundancy removal, feature extraction, data compression, etc. Because PCA is a classical technique which can do something in the linear domain, applications having linear models are suitable, such as image processing. Face recognition has many applicable areas. Moreover, it can be categorized into face identification, face classification, or sex determination. The main idea of using PCA for face recognition is to express the large 1-D vector of pixels constructed from 2-D facial image into the compact principal components of the feature space.

FASTMAP:

A new preprocessing algorithm, FastMap, is presented for embedding the nodes of a given edge-weighted undirected graph into a Euclidean space. In this space, the Euclidean distance between any two nodes approximates the length of the shortest path between them in the given graph. Later, at runtime, a shortest path between any two nodes can be computed using A* search with the Euclidean distances as heuristic estimates. FastMap is orders of magnitude faster than competing approaches that produce a Euclidean embedding using Semidefinite Programming. FastMap algorithm also produces admissible and consistent heuristics and therefore guarantees the generation of optimal paths.

GROUP MEMBER

Jung-Kang, Su	2389753352	Research for application; package comparison; edition
Yuhsi Chou	6048573191	Create PCA1.py and FastMap1.py; edition
Zhang Mujie	2621330761	Create PCA_FastMap.py

REFERENCE

<http://www.theanalysisfactor.com/tips-principal-component-analysis/>
<http://gromgull.net/blog/category/machine-learning/>
<https://zh.wikipedia.org/wiki/%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90>
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.6655&rep=rep1&type=pdf>
<https://arxiv.org/abs/1706.02792>