INF 552: Assignment 3
Dimensionality Reduction

Priyambada Jain (priyambj@usc.edu)
Sai Sree Kamineni (skaminen@usc.edu)
Varad Kulkarni (vdkulkar@usc.edu)

Part 1: Implementation

Language used – Python 2.7

1) **PCA**

Command to run – python PCA.py *path of pca-data.txt*

python PCA.py pca-data.txt

Data Structures Used: Numpy array for data points

Result: Output file "*PCA_output.txt*" has 2D coordinates of the points.

First Principal Component Direction X
Second Principal Component Direction Y

Code Level Optimizations:

- numpy.linalg.eig() to calculate eigen values & eigen vector.
- Numpy.hstack() to create new matrix with selected eigen vector coordinates.

Algorithm:

1. Read data points and calculate covariance matrix
2. Calcualte eigen values & eigen vector
3. Find the 2 largest eigen values & change the eigen vector matrix by removing vector corresponding to least eigen valued direction.
4. With the new eigen matrix, compute new coordinates by applying dot product.

Challenges Faced:
Data storage in array and keeping note of dimensions of array was tough.

2) **FastMap**

Command to run – python fastmap.py *path of* fastmap-data*.txt path of fastmap-wordlist.txt*

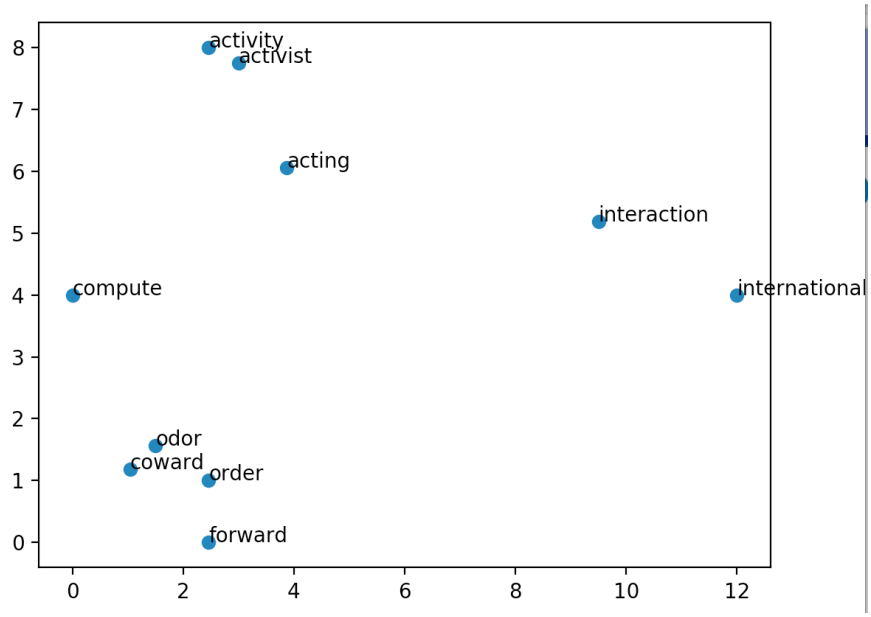
python fastmap.py fastmap-data.txt fastmap-wordlist.txt

Data Structures Used: Numpy array for data points

Results:
Final 2D points -
array([[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.]])

Plot of words in 2D



Code Level Optimizations:

- Math.sqrt to find square root in calculating distance
- Matplotlib.pyplot.subplots to plot the final points

Algorithm:

1. Start with k = n, where n is final space dimensions.
2. Initially choose farthest points (OA, OB)
3. For every new point (i) calculate the distance

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

4. Update this distance and repeat the process till k>0

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

Algorithm 2 *FastMap*
begin
Global variables:
$N \times k$ array **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 **PA**[]
/* stores the ids of the pivot objects - one pair per recursive call */
int col# =0;
/* points to the column of the **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 */
PA[1, col#] = a; **PA**[2, col#]= b;
4) if ($\mathcal{D}(O_a, O_b) = 0$)
set **X**[i, 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

Challenges Faced:
Coming up with and algorithm to implement was hard.
Finding the farthest pair considering time complexity was challenging.

Part 2: Software Familiarization

1) PCA

Python sklearn has an implementation for PCA.
sklearn.decomposition.PCA function:
pca = PCA(n_components=2)
pca.fit(input data)

Comparing the results

Feature	Code	Sklearn
Covariance	[[81.23061223 -15.83982766 31.66606691] [-15.83982766 13.70079979 -15.26416298] [31.66606691 -15.26416298 31.36555593]]	[[81.22845778 -15.83817402 31.66312677] [-15.83817402 13.69953054 -15.26190629] [31.66312677 -15.26190629 31.36154358]]
Matrix from eigen vectors	[[0.8666528 -0.49630987] [-0.23278176 -0.49246586] [0.44127722 0.71495027]]	[[0.86667137 -0.4962773] [-0.23276482 -0.4924792] [0.44124968 0.71496368]]
First data point in 2D	[10.9534134 7.41334173]	[10.95314032 7.41375984]

2) **FastMap**

Out of all the available implementations of FastMap in different languages, the following implementation is the better one.
Source code - https://github.com/mahmoudimus/pyfastmap
Levenshtein package required. Installation - pip install python-levenshtein
Command to run the code – python __init__.py
Note – The source code is included as part of submission. Before running the command, cd into the directory.

Part 3: Applications

1) PCA
   - *Email Classification* – Emails are part of day-to-day life. As the usage increases fraudulent activities increase as well. So the emails should be classified before reaching the user. Emails are classified by the words in it. Of all the words, few words like credit, pay, lottery etc imply fraud. So PCA is applied to the words (dimensions).
   - *Neuroscience* – Neural responses of stimulus has many dimensions out of which very few are relevant. Maximally informative dimensions is a variation of PCA used to reduce the dimensions to *k* relevant subspaces.
   - *Taxonomy* – A number of unweighted characters can be analyzed to group plant and animal species. PCA helps reduce the number of possible groupings. It replaces the original characters with only significant ones.

2) FastMap
   - *Data Clustering* – Data in N dimensions is reduced to k dimensions using FastMap and objects are projected onto k-dimensional space. Clustering algorithm is applied in low dimensional space.

- *Cluster Validation* – Clustering is performed in original space and FastMap is used to project clusters on low dimensional space and visualize them. If the cluster appears to be separate from other objects on the low dimensional plots, then it can be claimed that the cluster is also separate from original objects.

References:

1) https://www.intechopen.com/books/principal-component-analysis-multidisciplinary-applications/application-of-pca-in-taxonomy-research-thrips-insecta-thysanoptera-as-a-model-group
2) http://dataconomy.com/2016/01/understanding-dimensionality-reduction/
3) https://en.wikipedia.org/wiki/Maximally_informative_dimensions
4) http://repository.cmu.edu/cgi/viewcontent.cgi?article=1577&context=compsci
5) http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
6) https://books.google.com/books?id=Dc9rCQAAQBAJ&pg=PA234&lpg=PA234&dq=fastmap+applications&source=bl&ots=Gos-txJRJv&sig=anyQPr_Y70WvGCC2TOPVkeOjnJo&hl=en&sa=X&ved=0ahUKEwjZ66Wi1uLWAhWixVQKHWdIDvkQ6AEITzAH - v=onepage&q=fastmap applications&f=false
7) https://github.com/mahmoudimus/pyfastmap

Appendix:

PCA.py – Implementation of PCA.
pca-data.txt – Input file for PCA.
PCA_output.txt – Output file for PCA.

fastmap.py – Implementation of FastMap.
fastmap-data.txt, fastmap-wordlist.txt – Input files for FastMap.
Figure_1.png – Plot of words after applying FastMap.

Library_pyfastmap-master – Directory containing source code for Fastmap implementation from web.

__init__.py – Main function
stringmap.py – FastMap code
fastmap-wordlist.txt – input for the source code
result.png – Plot after running the script
README – Documentation of the code