

Programming Assignment 3 Report

PART 1: IMPLEMENTATION

PCA Algorithm:

Data Structures:

- **NumPy arrays:** I have used numpy arrays throughout the without libraries program because of their ease of usage and their efficient memory storage. I have used them as the data structures for storing data related to datapoints, covariances, eigenvalues and their sorted values, eigenvectors and their sorted values, and for storing the mean normalized values. I have also used numpy arrays for storing the final solution which is the reduced data from 3-dimensions to 2-dimensions.
- **Numpy.ndarray and numpy.matrix:** I have used loadtxt module to read the data in the with libraries implementation. This returned the data in the form of a numpy.ndarray. I have further converted this into numpy.matrix by using numpy.mat program for ease of computation.

Code-level optimizations:

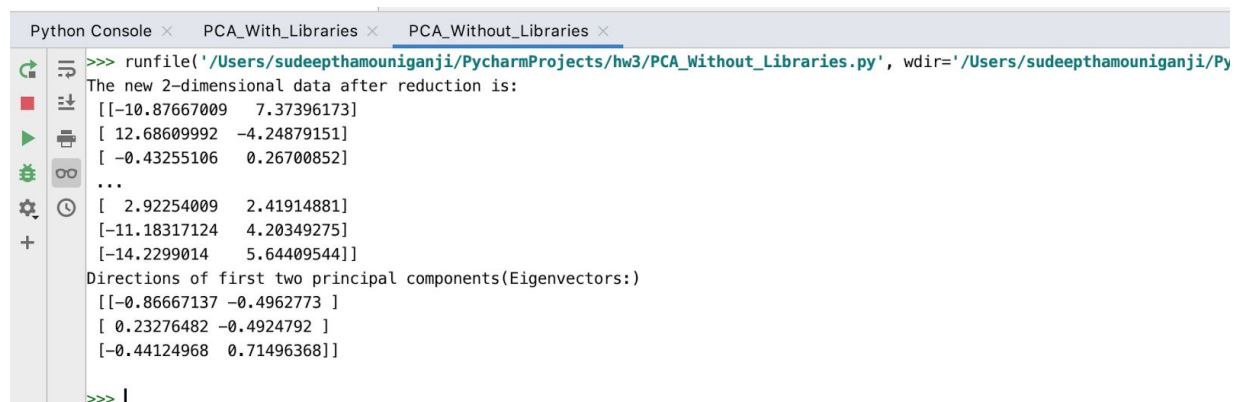
- I have extensively used numpy arrays instead of lists or other data types in order to improve the efficiency of the program.
- I have divided the program into different modules in order to improve the readability of the program. This would help anyone to read and understand the algorithm easily.
- I have used plots to graphically represent the obtained data for representation, comparison and understanding purposes.
- I have performed mean normalization of data so that the standard deviation is reduced and all the data points can contribute equally to the algorithm for the calculation of relevant axes.

Challenges:

- Initially, the values weren't exactly the same in both the implementations. But by performing, mean normalization I was able to achieve accurate outputs.
- We also had some confusion regarding how to get the first two principal components' directions. After some research on the matter, we decided to sort the obtained eigenvalues and eigenvectors which helped determine our final output.

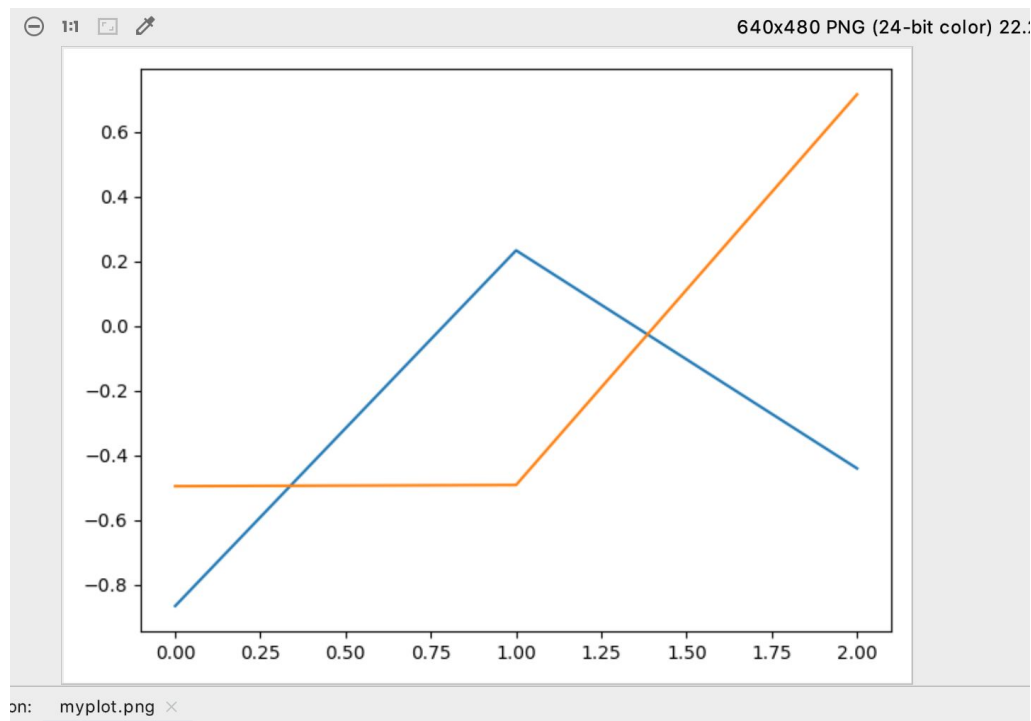
Output: The direction of the first two principal components(eigenvectors) without using libraries is:

```
[[ -0.86667137 -0.4962773 ]
 [  0.23276482 -0.4924792 ]
 [ -0.44124968  0.71496368]]
```



```
Python Console × PCA_With_Libraries × PCA_Without_Libraries ×
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw3/PCA_Without_Libraries.py', wdir='/Users/sudeepthamouniganji/Py
The new 2-dimensional data after reduction is:
[[-10.87667009  7.37396173]
 [ 12.68609992 -4.24879151]
 [-0.43255106  0.26700852]
 ...
 [ 2.92254009  2.41914881]
 [-11.18317124  4.20349275]
 [-14.2299014  5.64409544]]
Directions of first two principal components(Eigenvectors:)
[[-0.86667137 -0.4962773 ]
 [ 0.23276482 -0.4924792 ]
 [-0.44124968  0.71496368]]
>>> |
```

- Below is the plot for the implementation without the libraries part.



FastMap Algorithm:

Data Structures:

- **Pandas Data Frame:** I originally used the Pandas data frame to read in the wordlist file and the fast map data with the objects and their distances. However, for the actual implementation of the FastMap algorithm, I converted the data frames into different data structures for easier implementation
- **List of lists:** I created a NxN matrix using a list of lists in order to hold the distances between the objects. I used the Pandas data frame to more easily read in the data into a list of lists. Each row and column index corresponds to the objects and their value in the matrix is their distance away from each other.
- **Dictionary of lists:** I used a dictionary of lists to store the results of the coordinates after each iteration of the FastMap algorithm. The key holds the index of the object/word and the value holds a list consisting of the coordinates of that object in the 2D plane.

Code-level optimizations:

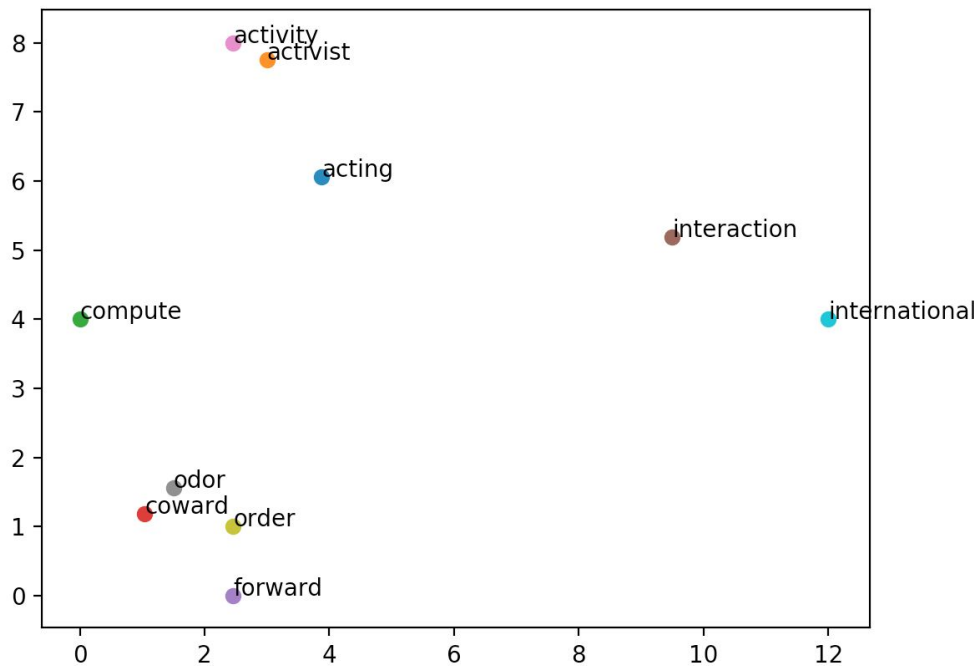
- I used the pivot changing heuristic to find the objects with the furthest distance in linear time.
- I have used plots to graphically represent the obtained data for representation, comparison and understanding purposes.
- I stored the distances between objects in a 2D matrix so that the row and column indices could be used interchangeably, making the distances more easily accessed.

Challenges faced:

- I initially had trouble with understanding how to store the distances between objects so that they could be updated later when the new triangular projection distances were calculated.
- I also was a bit confused on how to store the point projections between multiple iterations of the algorithm.

Output:

- Plot of the words in a 2D plane



- Coordinates of the words in a 2D plane
 - 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, 4.0]

PART 2: SOFTWARE FAMILIARIZATION

PCA Algorithm

Libraries

- **Numpy:** I have used NumPy library for reading the input data and converting it to a matrix by using its loadtxt function and mat function. Numpy also has a built-in function called mean which can be used to easily and efficiently find the mean of any data. I have used the mean function to find the mean of the input data so as to modify the input data before passing it to the PCA function.
- **Sklearn.decomposition:** sklearn.decomposition has many matrix decomposition algorithms like PCA, ICA, etc. Most of the algorithms in this module are regarded as dimensionality reduction algorithms. I have used the PCA algorithm from this library module to perform dimensionality reduction and for finding out the eigenvectors. This was done by fitting the modified data to the

PCA module and then using `pca.components_` to get the eigenvectors values in the projection space.

- **Matplotlib:** I have used matplotlib library to plot the output for the libraries implementation and without libraries implementation by using pyplot module.

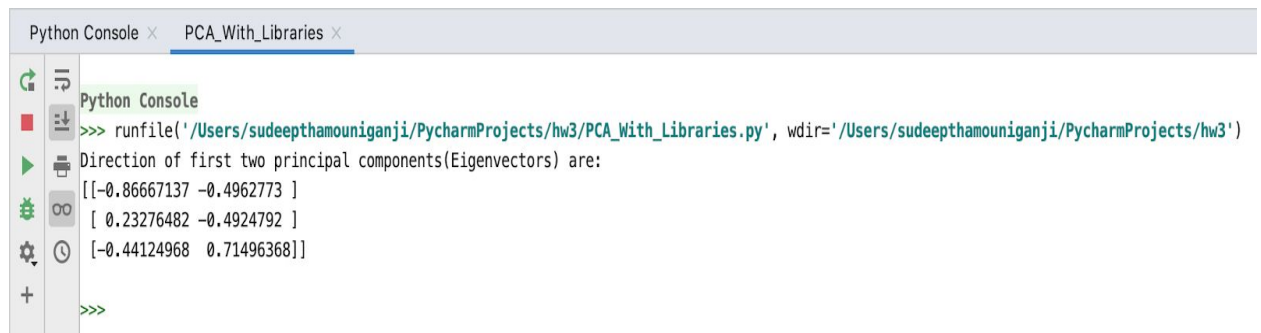
Implementation Comparison

- The in-built libraries in Python provide a very easy method of implementation of the PCA algorithm. Other than the PCA algorithm, `sklearn.decomposition` module has many other algorithms as well which help to achieve dimension reduction efficiently.
- In my without libraries implementation, on making several changes and using numpy's array data structure and in-built mean and covariance functions I have been able to efficiently get an accurate solution whose values were similar to that of the library implementation.

Result Comparison

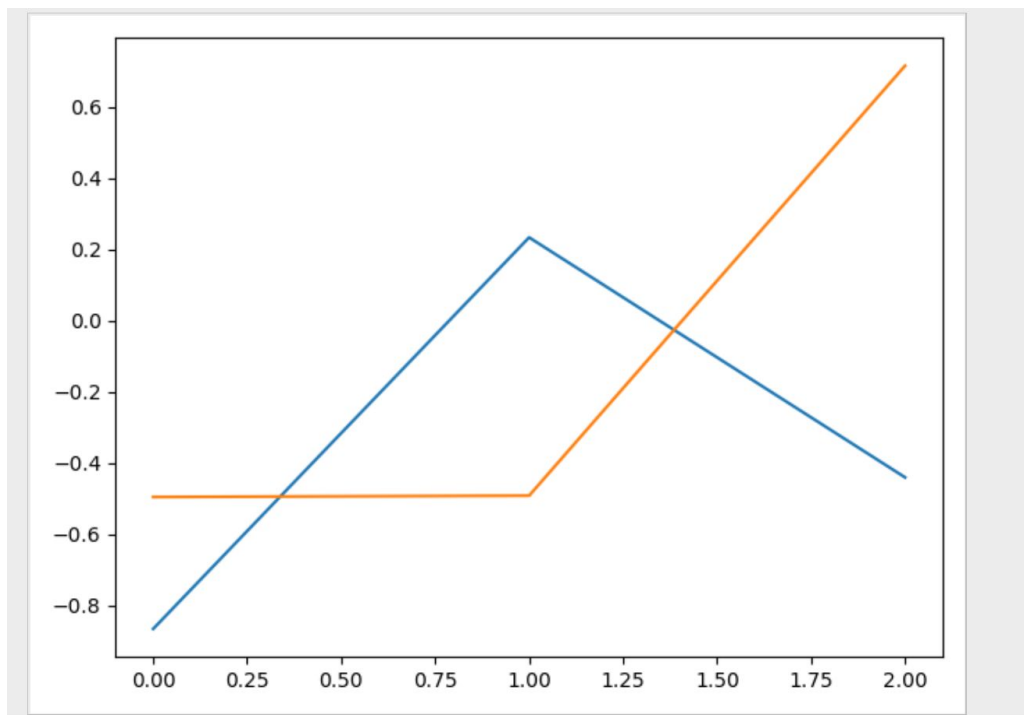
- **Library implementation output:**

```
[[ -0.86667137 -0.4962773 ]  
 [  0.23276482 -0.4924792 ]  
 [-0.44124968  0.71496368]]
```



```
Python Console x PCA_With_Libraries x  
Python Console  
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw3/PCA_With_Libraries.py', wdir='/Users/sudeepthamouniganji/PycharmProjects/hw3')  
Direction of first two principal components(Eigenvectors) are:  
[[ -0.86667137 -0.4962773 ]  
 [  0.23276482 -0.4924792 ]  
 [-0.44124968  0.71496368]]  
>>>
```

- The eigenvectors(direction of first two principal components) obtained in both the implementations were similar.
- Below is the plot for the library implementation part.



- The plots of the directions of the first two principal components also were similar for both the implementations.

Possible Improvements

- My implementation is slightly verbose and time taking when compared to the library implementation. We could make improvements in the code in order to make it run faster on larger datasets. However, the outputs obtained are the same.
- The library implementation allows for specifying how the svd_solver could be used and whether it should use a LAPACK implementation of the full SVD or a randomized truncated SVD. I think this would be a good addition to my implementation, especially when it is dealing with varied datasets.
- Additionally, with PCA algorithm, we could sometimes lose discriminating data and noise. We could add a possible extension to PCA to improve it and store such data separately.

FastMap Algorithm

Libraries

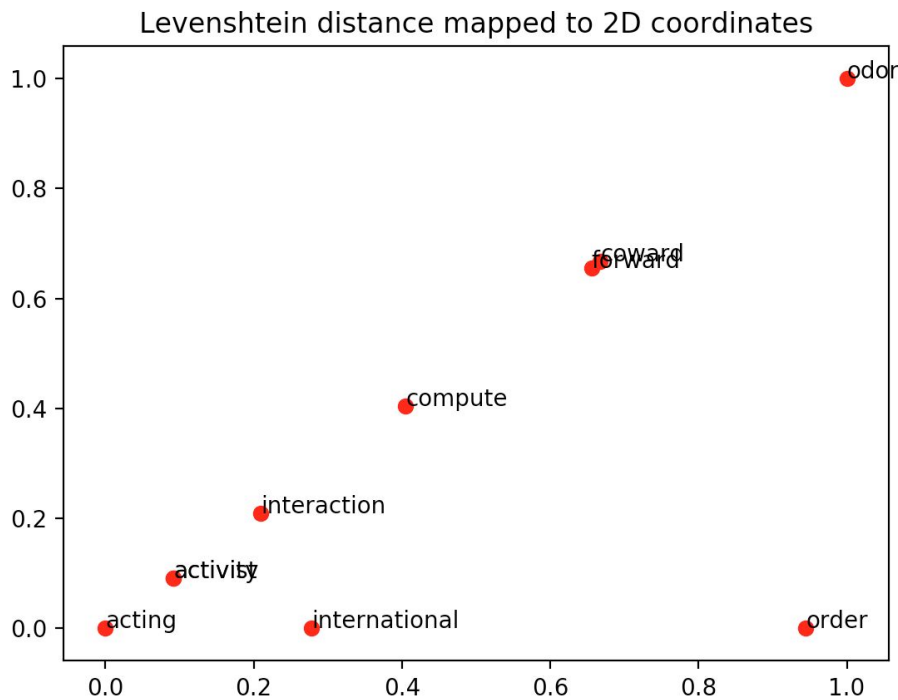
- **Gunnar Aastrand Grimes** (<http://gromgull.net/2009/08/fastmap.py>): Since there were no official Python library implementations of the FastMap algorithm, I utilized the one implemented by Gunnar Aastrand Grimes. His code was made available under the BSD license.
 - Libraries used: scipy, Levenshtein, math
- **Matplotlib**: I have used matplotlib library to plot the output for the libraries implementation and without libraries implementation by using pyplot module.

Implementation Comparison

- The FastMap algorithm implementation by Gunnar Aastrand Grimes uses the Levenshtein library. It utilizes the Levenshtein ratio which is the (Levenshtein distance / alignment length). The Levenshtein distance is a string metric for measuring the difference between two sequences.

Result Comparison

- Plot of the words in a 2D plane



- Coordinates of the words in a 2D plane
 - acting = [0.0 , 0.0]
 - activist = [0.09183673469387754 , 0.09183673469387754]
 - compute = [0.40493422661254835 , 0.40493422661254835]
 - coward = [0.6672222222222224 , 0.6672222222222224]
 - forward = [0.6555088268374981 , 0.6555088268374981]
 - interaction = [0.2092195309496347 , 0.2092195309496347]
 - activity = [0.09183673469387754 , 0.09183673469387754]
 - odor = [1.0 , 1.0]
 - order = [0.9444444444444444 , 0.0]
 - international = [0.27831667129944704 , 0.0]

Possible Improvements

- The result of the point projections in a 2D plane may differ from our implementation without a library because Gunnar Aastrand Grime's implementation uses the Levenshtein ratio in the distance matrix, while we used the distances from the fastmap-data.txt file. One possible improvement could be to use the scipy library like he did in order to store the results more easily.

PART 3: APPLICATIONS

PCA Algorithm

- **Facial Recognition Software:** Facial recognition software uses PCA with weighted eigenvalues to recognize faces. Here, every image is represented as a linear combination of weighted eigenvectors which are obtained from a covariance matrix of a training image dataset. The weights are determined based on a set of most relevant eigenvectors. These eigenvectors are called as eigenfaces. Facial recognition is done by projecting a test image into the subspace contained by eigenfaces while the classification is done by measuring minimum Euclidean distance.
- **Improved Raman Technique:** In nanotechnology, Raman spectroscopy is used to provide a structural fingerprint by which molecules can be identified and their quality can be determined. However, it also tends to harm the materials while it is testing. By using PCA, the Raman method has been improved to work at least 50 times faster and also rendered it gentler to work on sensitive materials. In this technique, PCA enhances the signal-to-noise ratio. It is used to establish the characteristics of noise and those of the real signals. However, the accuracy depends on the size of the dataset. If the dataset is larger, the recognition is more dependable and the actual signal can be differentiated more clearly.
- **Black hole images and movies:** The first-ever black hole image was obtained by using ML algorithms which were based on PCA without which getting the image of a black hole would have been impossible. PCA was used to characterize the high-fidelity simulations and interferometric observations of the millimeter emission that originates near the horizons of accreting black holes. The Fourier transforms of Eigen images derived from PCA were applied to an ensemble of images in the spatial domain to identify the black hole. Similarly, for making a movie about the black hole, it would require getting information about what happens in and around the black hole using ML algorithms. Here, the PCA algorithm will be learning about the black hole through the data which will be fed and then make a dimensionality reduction of that data which would act as the building blocks for the black hole images.

FastMap Algorithm

- **Human action recognition:** The automatic recognition of human actions is an ongoing challenge in the field of computer vision. Various factors must be considered like viewpoint variations, movement variability of similar actions, background interference, occlusions, and ambiguity of different actions. The FastMap dimensionality reduction technique can be used to embed a sequence of raw moving silhouettes to characterize the spatial-temporal property of the action. This could help to keep track of the geometric structure and provide a simple and fast recognition model.
- **Shortest path computation:** The FastMap algorithm can be used to embed nodes of an undirected edge-weighted graph into Euclidean space. The shortest path can then be calculated with A* search using the Euclidean distances as a heuristic. The FastMap algorithm applies to general undirected graphs as well and can be used with different heuristics.
- **Visual cluster validation in Data Mining:** Clusters can first be generated with general clustering algorithms from a database. The FastMap algorithm can then be used to project the clusters in a 2-dimensional or 3-dimensional space for visualization with different colors. This makes it easier for humans to visually examine the separation of clusters.

Group Members and Contributions

- Our group members for this programming assignment were Sudeeptha Mouni Ganji (ID: 2942771049) and Vanessa Tan (ID: 4233243951).
- Vanessa and Sudeeptha worked together to research and understand the algorithms.
- After implementation, Sudeeptha and Vanessa worked together to write the report.

Programming:

PCA algorithm <ul style="list-style-type: none"> • Without libraries • With libraries 	Sudeeptha Mouni Ganji Sudeeptha Mouni Ganji
FastMap algorithm <ul style="list-style-type: none"> • Without libraries • With libraries 	Vanessa Tan Vanessa Tan

Report Writing:

Part 1: <ul style="list-style-type: none"> • PCA algorithm • FastMap algorithm 	Sudeeptha Mouni Ganji Vanessa Tan
Part 2: <ul style="list-style-type: none"> • PCA algorithm • FastMap algorithm 	Sudeeptha Mouni Ganji Vanessa Tan
Part 3: <ul style="list-style-type: none"> • PCA algorithm • FastMap algorithm 	Sudeeptha Mouni Ganji Vanessa Tan