Lisa Meng
Shagun Gupta
INF 522
Homework 3

REPORT

Part 1: Implementation

1. Implementation of PCA from scratch
    a. Data Structure: Throughout the implementation of PCA from scratch, the data structures used were numpy array and Pandas DataFrame.
        i. The data file, "'pca-data.txt", was imported as a numpy array.
        ii. But then converted to a Pandas DataFrame for better visualization and to add column titles. To normalize the dataset, the Pandas df.mean() and df.subtract() functions were used to calculate the mean of each column and the data points were subtracted by their respective column mean. The df.cov() function was then used on the new (normalized) dataframe to calculate the covariance matrix, outputting as a Pandas dataframe.
        iii. To calculate the eigenvalues and eigenvectors, we used the numpy function "np.linalg.eig(cov_matrix)", which outputted a numpy array as the data structure. After sorting the eigenvectors to their respective eigenvalues in descending order, the numpy .delete() function was used to truncate the eigenvector matrix to n x k dimensions. Finally, we projected the transposed truncated eigenvector matrix to the normalized data points by dot product, thus reducing the dimensions from 3D to 2D, to form principal components, outputting as a numpy array.
        iv. We then converted the numpy array of the first 2 principal components into a Pandas DataFrame for better visualization and input it into a matplotlib graph. (Output of principal components and graph is in Outputs section below)

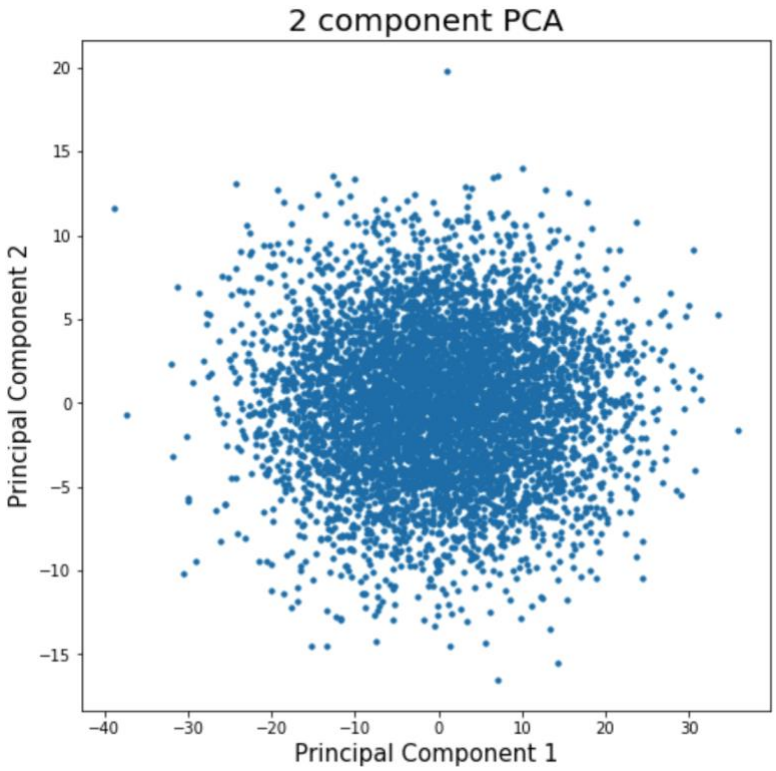
    b. Code-Level Optimization: The main way we optimized our code was using basic arithmetic operations (mean and subtraction) methods/functions that are able to be performed by Pandas or numpy, rather manually coding loops with "+, /, -, etc." operators. For instance, as stated above, to normalize our dataset, we were able to use the .mean() and .subtract() functions from Pandas to calculate the mean of *each* column then subtract *each* data point by their *respective* column mean within 2 lines of code. In addition, we used the np.linalg.eig(cov_matrix) from numpy to calculate the eigenvalues and eigenvectors of our covariance matrix which greatly simplified our code.

c. Challenges: Our greatest challenge was determining which way to “normalize” our dataset. When we researched which library to use for Part 2, we noticed that there were many different methods, such as standard scaling (sklearn.preprocessing.StandardScaler), normalizing (sklearn.preprocessing.normalize(data)), and scaling (sklearn.preprocessing.scale(data)). But ultimately, we declined to normalize our data according to what the professor taught in class: by subtracting the mean from each of the data dimensions, because the raw data did not have a wide range (i.e. the data points only range in the tens).

d. Outputs:
   i. The directions of the first two principal components (for first 10 data points)

	principal component 1	principal component 2
0	10.876670	7.373962
1	-12.686100	-4.248792
2	0.432551	0.267009
3	-6.192442	-0.369839
4	13.482977	-1.655150
5	-0.094842	-1.909424
6	-18.597238	-4.435651
7	8.133362	0.140160
8	7.178859	0.990978
9	-16.431421	-7.307489
10	2.278511	10.895239

   ii. Visualization of the first two principal components



2. Implementation of FastMap from scratch
    a. Data Structure: The implementation of fastmap was done using pandas DataFrame and lists.
        i. Pandas dataframe was used to import the fastmap wordlist and distance data. It was also used to plot the final output and store the new embeddings.
        ii. Lists were used within functions as parameters as well as code level implementation.
        iii. The data did not require any cleaning or handling. In fact, data structures did not play any integral role in the implementation of the algorithm as a data matrix was not to be maintained.
    b. Code-Level Optimization:
        i. To calculate the distance between 2 objects a distance function was called and distance was not stored in any distance matrix. This limited the number of calls to the dataframe and helped in implementing the algorithm in linear time (as required). The distance function took the original data file and subsequently subtracted the new embedding points thus not changing the data at all nor storing the changes anywhere.
        ii. To calculate the farthest points another linear time function is used, where a point is picked randomly and the farthest point is found from it. Then this point is the new point of center and the farthest point is found from it.

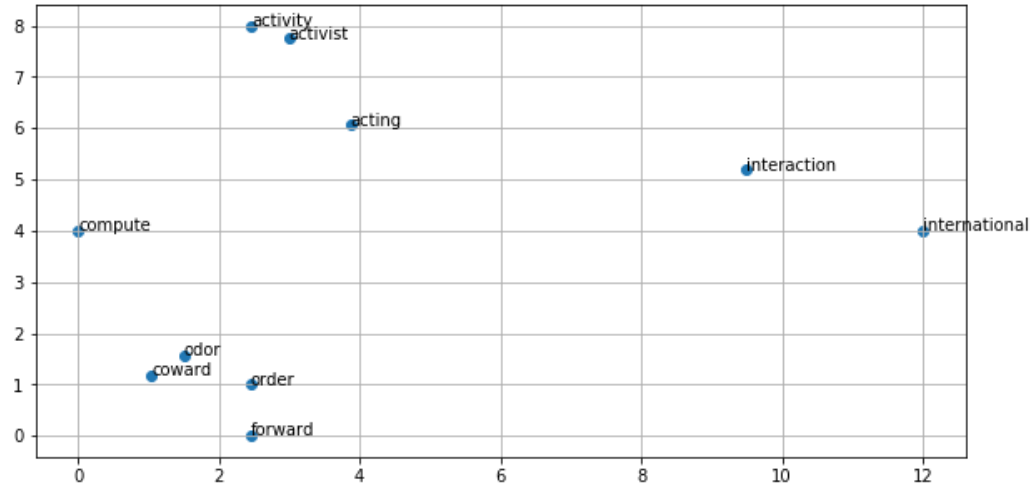
This is repeated till the new point found is the same as the previous one. This takes linear time as well(as explained in the lectures).

c. Challenges:
    i. Due to the complexity of the algorithm, limited tools to visualise it (can not visualise the objects to check if the embedding is correct) as well as the lack of proper libraries, it was hard to debug and check if the algorithm was correct or not initially.
    ii. It was also a little confusing to transfer the data into the next iteration without calling the entire dataframe but was resolved eventually.

d. Outputs:
    i. The embedding

	0	1	2
0	3.875000	6.0625	acting
1	3.000000	7.7500	activist
2	0.000000	4.0000	compute
3	1.041667	1.1875	coward
4	2.458333	0.0000	forward
5	9.500000	5.1875	interaction
6	2.458333	8.0000	activity
7	1.500000	1.5625	odor
8	2.458333	1.0000	order
9	12.000000	4.0000	international

    ii. The graph of the embedding

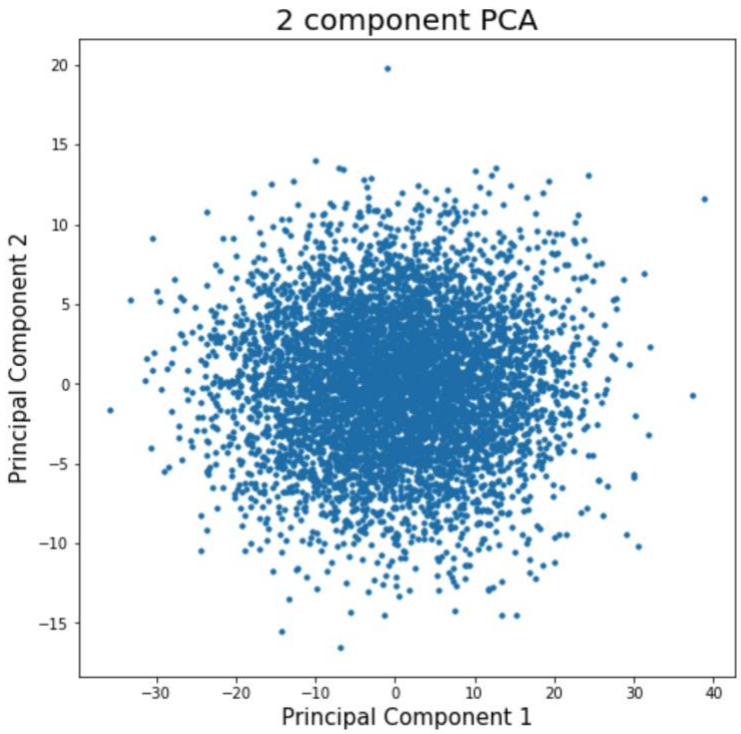


Part 2: Software Familiarization

1. Implementation of PCA using library

Upon researching libraries for PCA, we discovered "sklearn" offered the best implementation. When comparing against our implementation from scratch, we realized that there were not much difference (although the use of high-level library functions made the code, of course, shorter) because we obtained the same results/outputs, as shown below.

	principal component 1	principal component 2
0	-10.876670	7.373962
1	12.686100	-4.248792
2	-0.432551	0.267009
3	6.192442	-0.369839
4	-13.482977	-1.655150
5	0.094842	-1.909424
6	18.597238	-4.435651
7	-8.133362	0.140160
8	-7.178859	0.990978
9	16.431421	-7.307489
10	-2.278511	10.895239



Note that the graph of the principal components from Part 1 (coding from scratch) (on page 3) is the same as the graph from the use of a library in Part 2 (above), but are exact mirror images because the signs of the eigenvectors can be either positive or negative and is dependent upon how it is calculated by specific functions. This can be easily remedied by multiplying by -1.

Because our code from scratch is only about 10 lines and returned the same answer as using a library, we believe that our code does not require further improvement in terms of effectiveness at dimensionality reduction and calculating the principal components. But we do feel that a great way to improve our code is to further research and understand the "normalization" process and how to determine which method to use, since this will affect the overall result the most.

Part 3: Applications

1. PCA is the mathematically correct algorithm for dimensionality reduction. Real-world applications of PCA are:
    a. Images: PCA is used in the eigenface method for facial recognition by reducing the number of variables. PCA does this by transforming faces into a small set of essential characteristics, eigenfaces, which are the features in the training set. Recognition is performed by projecting a new image in the eigenface subspace, which then classification takes place. Another example is image compression, where PCA is used to remove less significant eigenvectors in order to decrease the size of the image for storage.
    b. Patterns: PCA is also used for finding patterns in data of high dimensionality in fields such as finance, data mining, and bioinformatics.

    c. Neuroscience: PCA is used in spike-triggered covariance analysis to identify the specific properties of a stimulus that increase a neuron's probability of generating an action potential. PCA is also used in a procedure called spike sorting, where it is used to differentiate neurons based in attributes of its action potential.

2. FastMap can also perform dimensionality reduction by setting a lower k but it is faster than PC. Although it is not an accurate algorithm (i.e. does not give precise/same answer every time), it enables clustering and a lot of other machine learning algorithms. Examples of real-world applications of FastMap are:
    a. Based on the paper that Professor Satish has co-authored, FastMap can be used to determine shortest path computations, which is useful for graphs.
    b. According to the creators (Christos Faloutsos and Ling-Ip Lin) of the original FastMap, its intended applications are indexing, data-mining, and visualization of traditional and multimedia datasets, as stated in the title of their published paper.

Contributions:

- For Part 1, Lisa and Shagun collectively discussed the concepts of both dimensionality reduction algorithms and decided to initially code the algorithms individually. From the individual work, we then picked the best code to submit, which was Lisa's code for the PCA algorithm and Shagun's code for the FastMap algorithm.
- Part 2 and 3 were researched and completed by Lisa.
- The report was compiled by Lisa and Shagun.