CS4125 Coursework C, Group 10

Thomas Bos
Delft University of Technology
Delft, Netherlands
t.c.bos@student.tudelft.nl

Daniël van Gelder
Delft University of Technology
Delft, Netherlands
d.vangelder-1@student.tudelft.nl

Jessie van Schijndel
Delft University of Technology
Delft, Netherlands
j.vanschijndel@student.tudelft.nl

Fig. 1: Example faces from the DrivFace dataset.

I. Principle Component Analysis

A. Datasets

The data used in the experiments for this task consist of two datasets: DrivFace [1] and the Gene Expression RNA-Seq dataset [2]. The first is a 606 instance dataset with 6400 variables representing the 80x80 pixels of images of driver faces. Some examples of faces can be seen in Figure 1. The latter is a bigger dataset of 801 instances with 20531 variables representing the gene expression levels of patients with different types of tumours.

B. The Algorithms Implemented

1) PCA: The first algorithm tested in this task is a straightforward implementation of Principle Component Analysis. Our implementation is mainly inspired by the slides from [3]. Given a dataset with `row-data` in $\mathbb{R}^{m \cdot n}$ where n is the number of features and m is the number of samples, this is computed as follows: (1) center the data (i.e. subtract the mean from each data point), (2) compute the covariance matrix of the centered data, (3) compute the eigenvalues and eigenvectors of the covariance matrix, (4) select the d eigenvectors with the highest eigenvalues, which are the basis vectors of the optimal affine subspace. In order to project a data point x onto the subspace we can use the following formula,

$$\sum_{i=1}^{d}((x - \bar{x}^T u_i)u_i) + \bar{x}$$

where $\bar{x}$ is the mean of the original data matrix and $u_1, ..., u_n$ are the basis vectors. We get the time and space complexity as follows. Centering the data can be done in $O(mn)$, constructing the covariance matrix takes $O(m^2n^2)$, and calculating the eigenvalues and eigenvectors can be done in $O(n^3)$ [4]. This gives a total complexity of $O(n^3)$. Similarly, the space complexity for centering is $O(mn + n)$, to calculate the covariance matrix is $O(n^2)$, and computing the eigenvalues is $O(n^2)$ which gives $O(mn + n^2)$ in total.

2) Kernelized PCA: When applying PCA on a row-data matrix X with m data points and n variables, it can become very costly to deal with the $n \cdot n$ covariance matrix when $n >> m$. We can perform kernelization by calculating the Gram matrix of the centered data ($G = XX^T$) instead which results in an $m \cdot m$ matrix [3]. Afterwards, we calculate the eigenvalues and eigenvectors of $\frac{1}{m}G$, select the d eigenvectors with the highest corresponding eigenvalues, and convert the eigenvectors into basis vectors by mapping the eigenvectors as in

$$u_i = \frac{1}{\lambda_i} X^T \phi_i$$

where λ_i and ϕ_i are the i-th eigenvalue and corresponding eigenvector [3]. For the time complexity, we know that calculating the Gram matrix takes $O(m^2n^2)$, calculating its covariance matrix takes $O(m^3)$ [4], calculating the eigenvectors of the covariance matrix takes $O(m^3)$ [4], and mapping the top d eigenvectors to basis vectors takes $O(dm^2n)$. This gives a total time complexity of $O(m^3 + dm^2n)$. If we assume $n >> m$, then this complexity comes down to $O(dm^2n)$ which is better than PCA. Similarly, the space complexity for centering is $O(mn + n)$, to calculate the covariance matrix is $O(m^2)$, computing the eigenvalues is $O(m^2)$, and mapping the eigenvectors to basis vectors takes $O(mn + m)$, which gives $O(mn + m^2)$ in total. Again, this is better than PCA if we assume $n >> m$.

C. Tests Performed

The first test performed is to calculate the reconstruction error (RE) of both algorithms. This error is calculated by summing the squared difference between each value of the original and the reconstructed data matrix. In formal terms, the RE is calculated by first calculating the PCA reconstruction, $\hat{X} = XUU^T$ [5], from row data matrix X and matrix U of which the columns are the basis vectors, and then calculating the Frobenius norm of $X - \hat{X}$. To assess the difference in time taken to compute the results, the runs of each method will be timed for each d. For comparison, the Matlab base PCA implementation will be shown in the analyses as well. In the end we will go over some of the eigenfaces generated by the methods and compare the reconstructions of the original PCA

All authors contributed equally to this work.

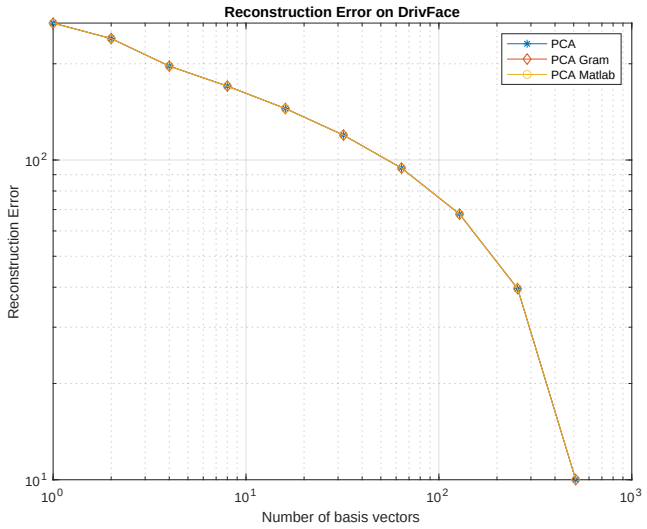


Fig. 2: The reconstruction error for the three implementations on the DrivFace dataset.

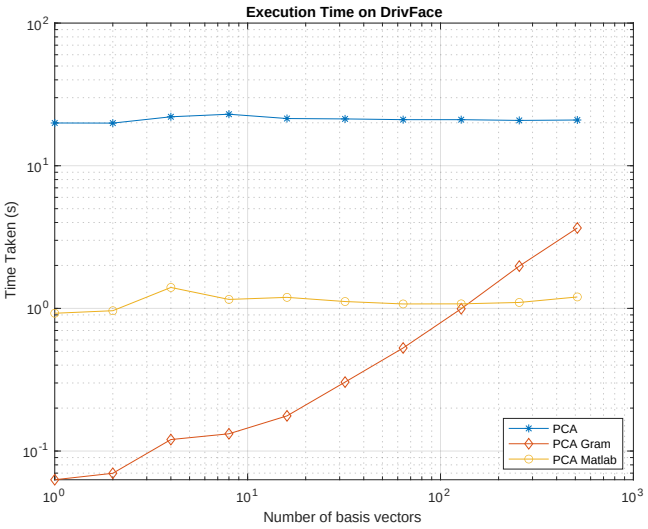


Fig. 4: Execution time for the three implementations on the DrivFace dataset.

with the reconstructions of Gram PCA for different values of d.

D. Results

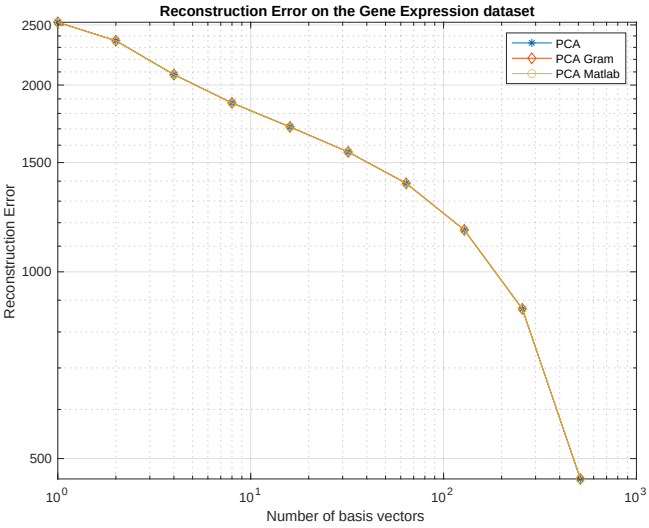


Fig. 3: The reconstruction error for the three implementations on the Gene Expression dataset.

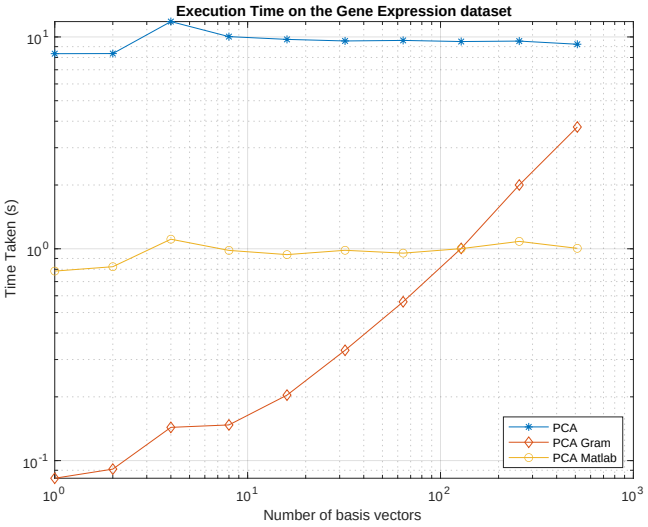


Fig. 5: Execution time for the three implementations on the Gene Expression dataset.

Figure 2 and 3 depict the Reconstruction Error computed for each method on the DrivFace and Gene Expression datasets. The number of basis vectors used varies between 1 and 512. We can clearly see that the error is the same for all three methods. The error decreases as more basis vectors are used to reconstruct the original data, which is to be expected as more of the original data is retained.

Figure 4 and 5 show the time it takes to compute the d basis vectors for each of the methods. As expected, the 'vanilla' PCA and Matlab PCA do not differ in computation time between themselves, as increasing d only changes how many eigenvectors are returned, but not the computations. The Matlab implementation uses less time likely due to optimizations. With the PCA Gram method, however, when d is increased more mappings between eigenvectors and basis vectors need to be done, increasing the computational time linearly in d. When the mappings of fewer than a hundred eigenvectors need to be computed, the PCA Gram method is even more time efficient than the Matlab PCA implementation.

We will now analyse some of the eigenfaces generated, reconstructions of DrivFace images 1, 101, and 201, and their respective reconstruction errors, which can be seen in Figure 6, 7, and I respectively. We can clearly make out facial features in the last three eigenfaces, indicating that PCA has successfully extracted them from the dataset. Furthermore, looking at the reconstructions, we can see that using more basis vectors

Image	$d = 606$	$d = 512$	$d = 128$	$d = 32$	$d = 8$	$d = 1$
1	2.81	2.68	5.69	9.32	14.94	16.24
101	2.79	2.75	5.07	9.14	12.46	14.39
201	2.70	2.77	4.81	8.26	11.99	15.21

TABLE I: Reconstruction errors for the images in Figure 7.

results in a better reconstruction, which is to be expected. Interestingly enough, reducing the amount of basis vectors for reconstruction results in the same image being reconstructed. We can explain this by the fact that the more basis vectors we leave out, the less facial features are captured, resulting in very similar images. If we look back to the RE calculated before, we can see that with $d = 1$ we get an error of 250 and visually very similar images. Increasing d to 128 yields an RE of $\tilde{7}0$ and we can already make out some of the features of the original face. Only until $d = 512$, when the error is reduced to $\tilde{1}0$, do the faces become clear. Also note that somewhere in the Gram PCA process some information was lost as even with 606 basis vectors the reconstructed image is still not the same as the original, which can be confirmed by looking at the errors which are greater than 0. We can conclude that in order for an image to be recognizable, its reconstruction error should be at least lower than 5.

Fig. 6: The first four eigenfaces generated by the PCA Gram method on the DrivFace dataset.

II. Fast Approximation of PCA

A. Datasets

Again, we utilize the DrivFace dataset described in the previous section.

B. The Algorithms Implemented

In the description of these algorithms, we assume column-based data (n x m) with m data points and n features.

Fig. 7: Gram PCA reconstructions for different values of d. The maximum d is 606 as that is the maximum dimension of the $n \cdot n$ Gram matrix.

1) Snapshot PCA: The Snapshot PCA algorithm is highly similar to the Kernelized PCA. However, rather than using all data, a sampled set of l points in the data is used. The new input data matrix thus becomes n x l. The rest of the computation is the same as the Kernelized PCA algorithm described earlier.

The time complexity of this algorithm is similar to the Kernelized PCA algorithm. However, instead of relying on m, the running time depends on l yielding a time complexity of: $O(dl^2n)$ assuming $l < m << n$ or $O(l^3 + dl^2n)$ otherwise.

2) Nyström Method: This approximation method uses sampling to extract "landmark" coordinates from the data. Consider the input data matrix to have n features for m data points. The landmark coordinates are selected by sampling l of the n features. The datamatrix is permuted such that the landmark coordinates are the first coordinates in the matrix. Then, the matrix is centered and of this centered matrix, the covariance matrix is constructed only for the landmark coordinates yielding an n x l matrix. This matrix is separated into a square l x l matrix A and an $(n-l)$ x l matrix B. The eigenvectors and eigenvalues of matrix A are computed. The eigenvectors are stored as columns in a matrix U_A and the eigenvalues are stored in a diagonal l x l matrix Λ_A. Then, the permuted PCA modes are given by:

$$\tilde{U} = \begin{bmatrix} U_A \\ BU_A(\Lambda_A)^{-1} \end{bmatrix}$$

The final PCA modes can be obtained by inverting the permutation performed previously on the matrix $\tilde{U}$.

The time complexity of this algorithm is as follows: permuting the data matrix (assuming the use of a permutation matrix for simplicity) is $O(n^2m)$, centering the matrix is $O(mn)$, calculating the covariance matrix $O(n^2l)$, calculating the eigenvectors of the square l x l matrix takes $O(l^3)$. Computing the permuted PCA modes takes $O(ndl)$ and finally, inverting the permutation takes $O(n^2m)$. Overall this yields a total time complexity of: $O(n^2m + n^2l + l^3)$.

C. Tests Performed

It is a challenge to evaluate the correctness of these approximation algorithms. However, by design, the Snapshot PCA and Nyström algorithms will be identical to PCA and Kernelized PCA, respectively, if the l parameter is set to its maximum value (i.e. $l = m$ for Snapshot PCA and $l = n$ for Nyström). Therefore we set up an experiment to compare the approximation algorithms to one of the original algorithms, as it has been previously demonstrated that these would yield identical PCA modes. If the approximation algorithms are indeed correctly implemented, then the PCA modes will be identical and the Reconstruction Error will be identical as well, given that the same dataset is used. In this case, we will perform PCA to 2 dimensions.

The second test that will be performed regarding the time/accuracy trade-off. As the l parameter is decreased, the runtime will decrease as well, as more approximation is taking place. On the other hand, the reconstruction error will increase

as well due to a higher level of approximation. Using the same dataset, an experiment is run where the entire DrivFace dataset is mapped down to 2 dimensions and mapped back again for varying values of l. Then, the runtime of the algorithms is measured as well as the sum of the reconstruction error for all datapoints in the dataset. To account for variance due to the random sampling, average values are taken over 5 runs.

Finally, to visualize the effect of varying l values a selected face of the DrivFace is mapped down to varying dimensions and using a given l and mapped back again.

While the reconstruction error is an effective measurement for accuracy, it should not be interpreted as an absolute value of performance. Instead, in these experiments we will consider it as a relative value of performance, to compare to other values to see an improvement/deterioration of performance.

These experiments will be concluded by a discussion of the results aiming to answer whether the approximation algorithms have been implemented correctly, as well as which algorithm is better to use in which scenario.

D. Results

Figure 8 shows results of the first experiment. The distribution of the points is identical. Note that for the Nyström scatterplot, the distribution seems to be flipped. This is due to the fact that some of the PCA modes are negative when compared to the PCA modes of the other algorithms. This could be due to the changing of the order of the coordinates by the permutation matrix, affecting the sign of the eigenvectors. The reconstruction error (for reconstructing the entire dataset) was identical as well, with a value of: 196.71. We can therefore conclude that when using maximum values of l the algorithms are identical to the non-approximation algorithms.

For the second experiment, the results are shown in figure 9. Observe that for both algorithms, as the value for l increases, the reconstruction error decreases *exponentially*. Whereas, the runtime seems to increase polynomially for both the Nyström and Snapshot PCA algorithms, although it is more difficult to determine the type of growth for the latter. Note that the variance of the runtime is large for the Snapshot PCA algorithm, but the runtime range of values is small as well ($<$ 1 second), explaining the larger variance. Overall, the results seem to confirm the theoretical runtime complexity described earlier.

Finally, the reconstructed faces of the third experiment are shown in figure 10. It is evident that for larger values of l and subspace dimension d, the quality of the reconstruction improves significantly. Note that the Nyström algorithm performs significantly worse, even when using a very large value for l. It is also interesting to note that for the lower subspace dimension ($d = 2$), the reconstructed face resembles the eigenfaces shown in figure 6. Overall, the results demonstrate that in order to provide a high-quality reconstruction, both the sampling parameter l and subspace dimension d need to be considered.

In conclusion, the results indicate that the algorithms have been implemented correctly, following the theoretical runtime complexity (Figure 8 and 9). Both algorithms are able to balance runtime and accuracy, although the Nyström method has a lower accuracy overall. However, the algorithms are suitable for varying use cases. Note that for these experiments, a dataset was used that had far less samples than features ($m << n$). As also evident from figure 9, the Snapshot PCA method is more suitable for a scenario where $m < n$ whereas the Nyström method would perform better in a scenario where $m > n$. On the other hand, the Nyström method is able to partially offset its worse runtime by sampling less landmark coordinates from the data. However, this may still yield a solution where the mapping is of lesser quality than using the Snapshot PCA. Therefore, it is better to apply the Nyström method in a data environment where there are more samples than features and the Snapshot PCA method in a data environment where there are more features than samples.

There is room for improvement in the implementation of the algorithms. The current sampling technique for both algorithms is random sampling without replacement. However, different techniques could be used to sample the points/coordinates. One technique that was attempted was to select the points with the highest variance in feature values. However, this did not yield a better result. A possible technique to explore could be to select the points with the highest average distance to the other points in the data, in order to capture as much of the feature space as possible. However, this has a substantial performance impact. A deeper investigation regarding performance improvements will be left as future work.

III. Multidimensional Scaling

A. Datasets

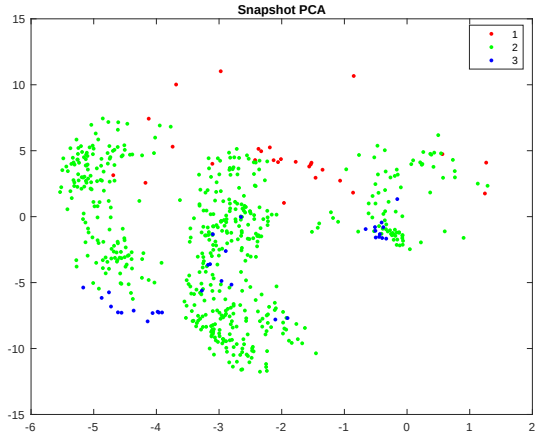
The datasets used in the experiment for this task are the Fisher Iris and the Ovarian Cancer datasets bundled with Matlab [6]. In addition, we used the MNIST handwritten digits dataset [7]. In order to parse the binary data of the dataset, we used a utility function provided by Siddharth Hegde [8].

B. Implemented Algorithm

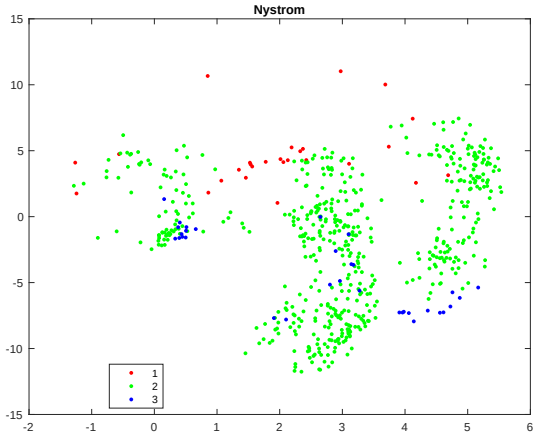
The algorithm implemented is the multidimensional scaling (MDS) algorithm. This algorithm may be used to transform high-dimensional data into a lower dimension. This dimension is referred to as d. The transformation aims to preserve between-object distances. The result can be used for visualization of the data. In our implementation, we first calculate the similarity matrix D in square form. We use two different distance metrics: the cosine distance and the spearman distance. These distances were chosen as they resulted in the best mappings. We then construct Gram matrix G from D using the double centering:

$$G = -\frac{1}{2}\left(I_n - \frac{1}{n}ee^T\right)D\left(I_n - \frac{1}{n}ee^T\right)$$

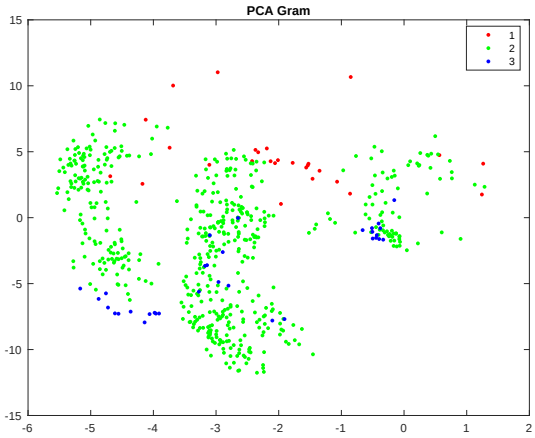
where I_n is the $n \times n$ identity matrix and e is a $n \times 1$ vector of ones. Then, we compute the eigenvalues $\boldsymbol{\lambda}$ and eigenvectors



(a) Results for the Snapshot PCA method. Parameter l was set to $m = 606$.



(b) Results for the Nyström method. Parameter l was set to $n = 6400$.



(c) Results for the Kernelized PCA method.

Fig. 8: Visualization of both PCA approximation algorithms and the Kernelized PCA algorithm using the DrivFace dataset reduced to 2 dimensions.

$\boldsymbol{u}$ of G. We select the d highest eigenvalues and corresponding eigenvectors. Then, we compute the points in matrix form:

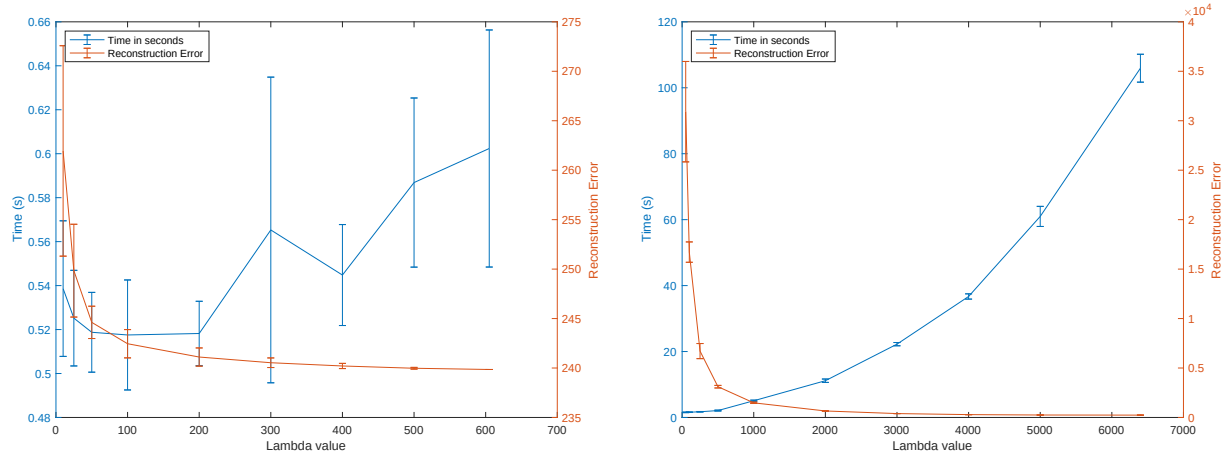
$$Y = (\sqrt{\boldsymbol{\lambda}}\boldsymbol{u})$$

C. Test Performed

To verify the reliability and validity, the stress is computed. In order to do this, we take the similarity matrix of Y. The resulting matrix contains the distances in the input matrix. Then, we subtract matrix D from these distances. Matrix D contains the actual distances. We square the result and sum the stresses to obtain the total stress s. The calculated total stresses are shown in Table II. We see that overall the stress is lowest for the Fisher Iris dataset and highest for the MNIST dataset. The cosine similarity metric shows the best performance.

D. Results

The calculated eigenvalues are shown in Table II. The visualization results of our experiments are shown in Figure 11, Figure 12 and Figure 13. For the Fisher Iris dataset we compare Figure 11a and Figure 11b. It can be seen that the cosine similarity performs better here. For the Ovarian Cancer dataset it is not clear which similarity metric performs better. While Figure 12a shows a clearly defined cluster for non-cancer observations, a large number of points from two clusters seem to be close together. In Figure 12b the cluster of non-cancer observations is less close together, but each cluster is more nicely distributed across the space. For the MNIST dataset the two mappings are visually quite similar, as can be seen from Figure 13a and Figure 13b.



(a) Results for the Snapshot PCA method.

(b) Results for the Nyström method.

Fig. 9: Runtime vs. Reconstruction Error for both PCA approximation algorithms for varying values of l while running on the DrivFace dataset. For each value of l, the average of 5 runs was taken, standard deviations are shown as well.

Distance metric	Dataset	Total stress s	Eigenvalues
Cosine	Ovarian Cancer	545.3372	2.9567 and 0.45359
Spearman	Ovarian Cancer	3139.783	4.1105 and 1.4189
Cosine	Fisher Iris	479.6306	3.2121 and 0.092075
Spearman	Fisher Iris	611.1456	6.6667 and 1.9586e-15
Cosine	MNIST	101692.5781	29.1432 and 23.0344
Spearman	MNIST	212803.1448	40.1277 and 33.7338

TABLE II: Total stress s and eigenvalues resulting from MDS for three datasets and two distance metrics

IV. Stochastic Neighbour Embedding

A. Datasets

For the comparative analysis we again use the Fisher Iris and the Ovarian Cancer datasets bundled with Matlab [6]. In addition, we used the MNIST handwritten digits dataset [7].

B. Implemented Algorithms

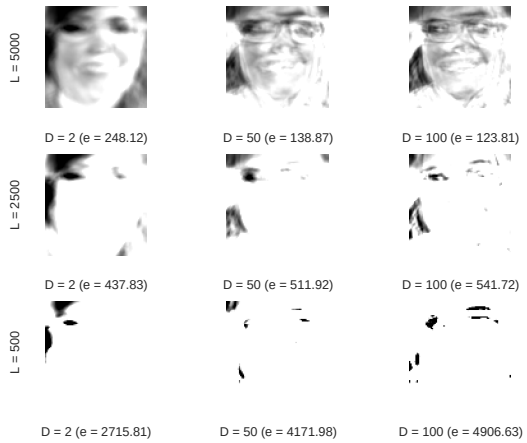
For this task we use the MDS function described in Section III and the tSNE function created by Matlab [1]. The name tSNE stands for t-distributed Stochastic Neighbor Embedding. The idea behind t-SNE is to adapt the probability distribution of the symmetric similarities such that points far away from each other are not so strongly affected. As the name suggests, it uses the student t-distribution for this purpose. Similar to the MDS algorithm, the t-SNE algorithm starts by calculating the similarity matrix D. By default, the matlab implementation uses the Euclidean distance. Then, it calculates a standard deviation σ_i for all data points in D. It calculates these standard deviations such that the perplexity of each row is equal to a specified perplexity. This perplexity is the effective number of neighbors for each point. Then, the similarity matrix is calculated. An initial low-dimensionality mapping is created and iteratively updated. The updates are performed such that the distance between a Gaussian distribution in the original dimensionality and a t-distribution in the low-dimensional space is minimized. We evaluate the same distance metrics as those used in section III. These metrics are the cosine similarity and the spearman similarity.

[1]Matlab's tSNE function: https://www.mathworks.com/help/stats/tsne.html

C. Results

The visualizations of the t-SNE mappings are shown in Figure 14, 15 and 16. These will be compared to the MDS mappings shown in Figure 11, Figure 12 and Figure 13. Figure 14 shows the t-SNE mappings for the Fisher Iris dataset. We compare this figure to Figure 11. The results are similar to that of MDS, although the clusters are more clearly separated in the t-SNE results for both similarity metrics. Figure **??** shows the t-SNE mappings for the Ovarian Cancer dataset. We compare this figure to Figure 12. We see different results for t-SNE than those for MDS. For t-SNE both similarity metrics produce more similar mappings than for MDS. Clusters are more easily separable in the t-SNE results. Figure 16 shows the t-SNE mappings for the MNIST dataset. Comparing this figure to Figure 13, the clusters are much better separated in Figure 16. Compared to MDS, t-SNE sacrifices relations between far away points and therefore some of the structure of the data in order to form tight clusters. Overall, this leads to improved clusters.

(a) Results for the Snapshot PCA method.



(b) Results for the Nyström method.

(c) Original face to reconstruct.

Fig. 10: Visualisation of reconstructions of a selected face from the DrivFace dataset by both approximation algorithms for varying levels of l (L) and subspace dimension d (D). Reconstruction errors are shown as well, denoted by "e".

REFERENCES

[1] UCI, "Drivface data set," https://archive.ics.uci.edu/ml/datasets/DrivFace#, 2016.

[2] ——, "gene expression cancer rna-seq data set," http://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq, 2016.

[3] K. Hildrebrandt, "Cs4125 seminar research methodology for data science (2020/21 q4), lecture 9," 2021.

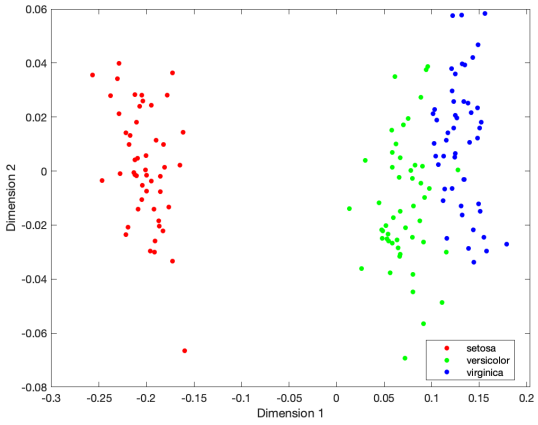
[4] T. Elgamal and M. Hefeeda, "Analysis of pca algorithms in distributed environments," 2015.

[5] amoeba, "How to reverse pca and reconstruct original variables from several principal components?" https://stats.stackexchange.com/questions/229092/how-to-reverse-pca-and-reconstruct-original-variables-from-several-principal-com, 2016.

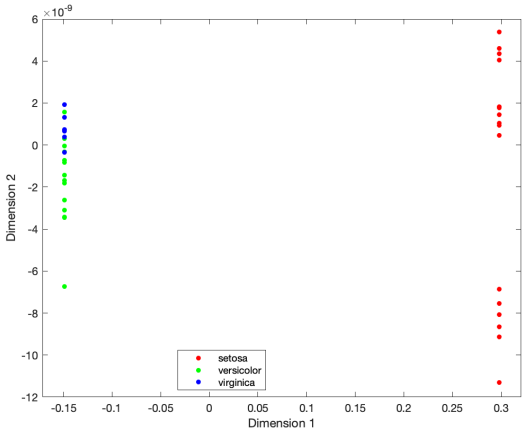
[6] MATLAB, *version 9.10.0 (R2021a)*. Natick, Massachusetts: The MathWorks Inc., 2021.

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[8] S. Hegde. Read digits and labels from mnist database. [Online]. Available: https://nl.mathworks.com/matlabcentral/fileexchange/27675-read-digits-and-labels-from-mnist-database

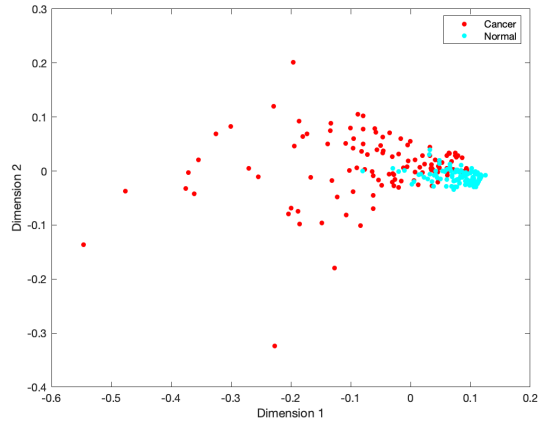


(a) Result of cosine similarity MDS for the Fisher Iris dataset

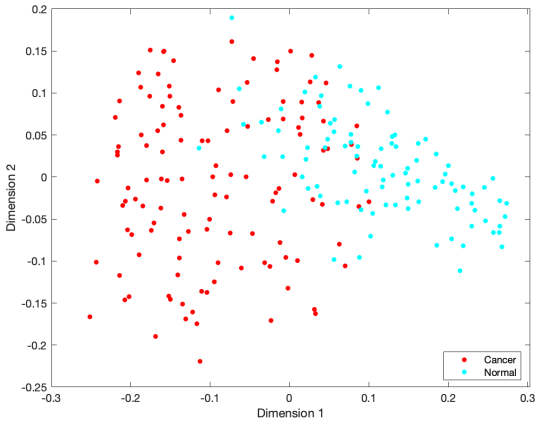


(b) Result of spearman similarity MDS for the Fisher Iris dataset

Fig. 11: Results of MDS for the Fisher Iris dataset

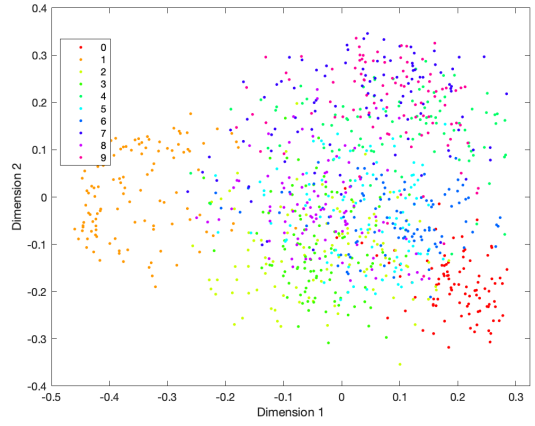


(a) Result of cosine similarity MDS for the Ovarian Cancer dataset

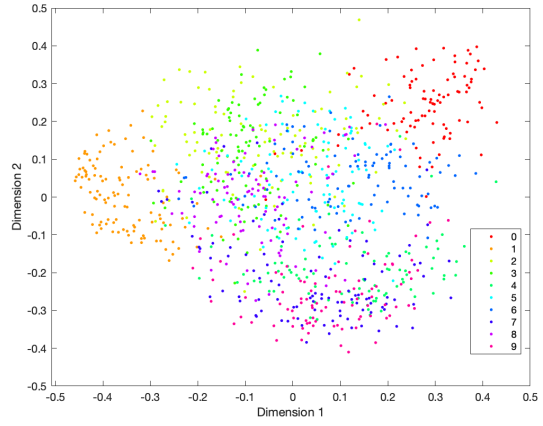


(b) Result of spearman similarity MDS for the Ovarian Cancer dataset

Fig. 12: Results of MDS for the Ovarian Cancer dataset

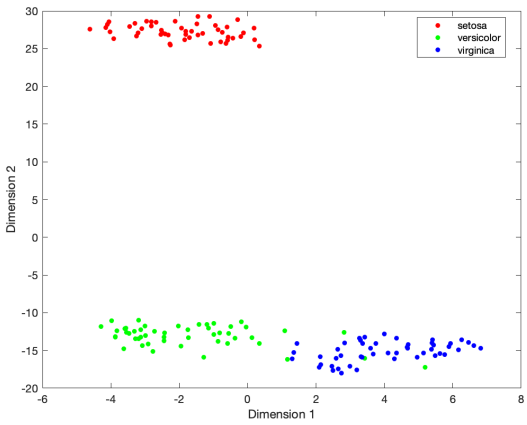


(a) Result of cosine similarity MDS for the MNIST dataset

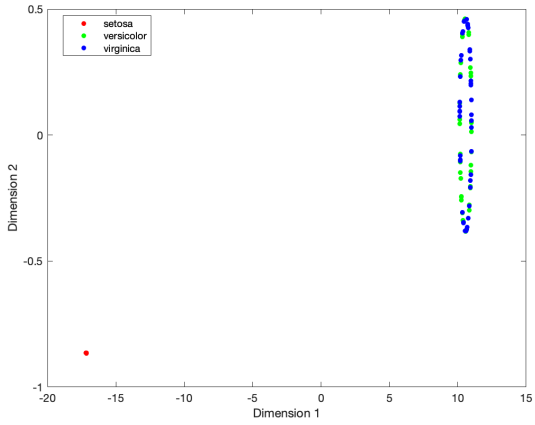


(b) Result of spearman similarity MDS for the MNIST dataset

Fig. 13: Results of MDS for the MNIST dataset

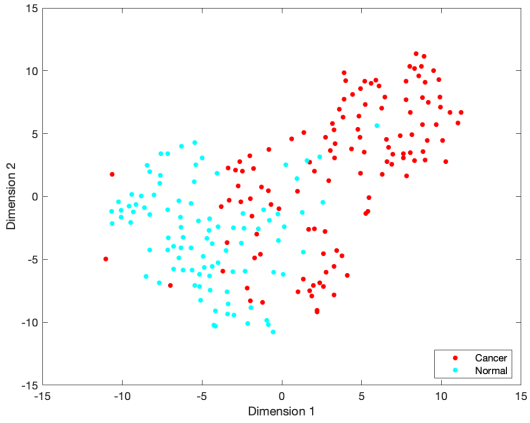


(a) Result of cosine similarity t-SNE for the Fisher Iris dataset

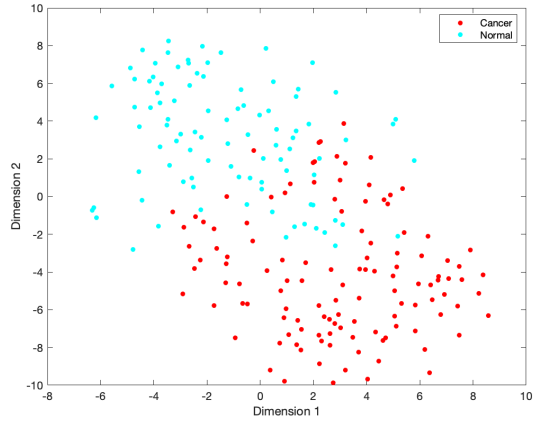


(b) Result of spearman similarity t-SNE for the Fisher Iris dataset

Fig. 14: Results of t-SNE for the Fisher Iris dataset

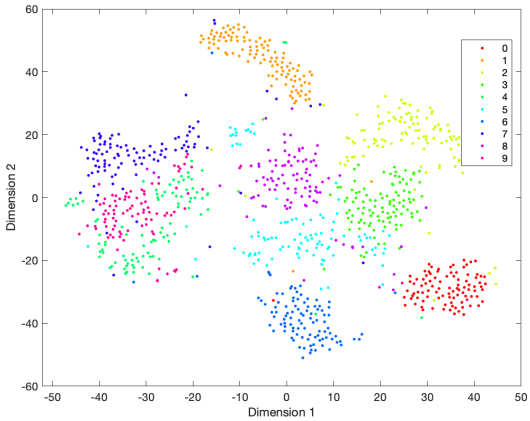


(a) Result of cosine similarity t-SNE for the Ovarian Cancer dataset

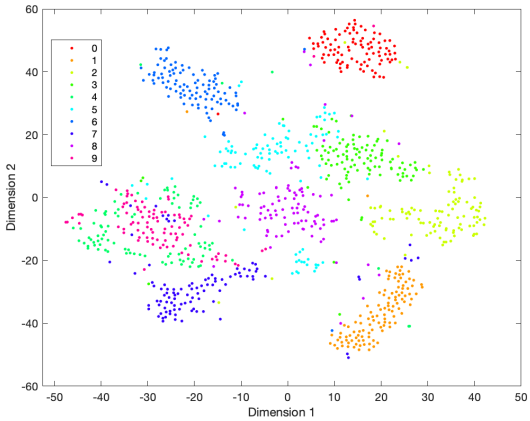


(b) Result of spearman similarity t-SNE for the Ovarian Cancer dataset

Fig. 15: Results of t-SNE for the Ovarian Cancer dataset



(a) Result of cosine similarity t-SNE for the MNIST dataset



(b) Result of spearman similarity t-SNE for the MNIST dataset

Fig. 16: Results of t-SNE for the MNIST dataset