Introduction to Machine Learning
Assignment 7

Group 06
Lucas Pereira (s4507983) & Matteo Wohlrapp (s4974921)

November 17, 2021

Introduction

In this practical, we experiment with principal component analysis, short PCA. PCA is a method mainly used for dimensionality reduction which aims to maximize the variance when projecting the data to lower dimensions. In general, dimensionality reduction maps data points from higher dimensions to lower dimensions: $f : R^n \rightarrow R^d, n >> d$. This procedure is used for various reasons. With lower dimensions, time and storage space for algorithms are reduced. It can also be used to reduce noise or extract features from the data. Furthermore, it helps to avoid the curse of dimensionality, which causes statistical methods to fail in higher dimensions. Finally, dimensionality reduction is also useful for data visualization.

Methods

The basic idea of PCA is to project the high dimensional data into low dimensional space using a linear function: $f : R^n \rightarrow R^d$ where $f(x) = W^T x$ and $W = [u_1, u_2, ..., u_n]$ is a matrix. PCA then defines W in a way which minimizes the information loss, or in other words maximizes the variance of the projected data. For our definition, we define the variance for a given set of observations $\{y_1, y_2, ..., y_n\}$ as $\sigma_y^2 = \frac{1}{M}\Sigma_{i=1}^{M}(y_i - \overline{y})^2$, where $\overline{y} = \frac{1}{M}\Sigma_{i=1}^{M} y_i$. Intuitively we are searching for solutions similar to the one shown in **Figure 1**.

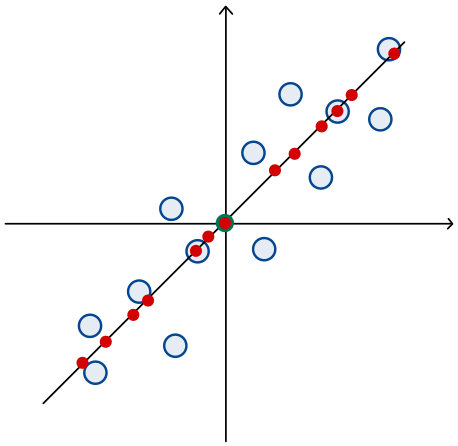

Figure 1: *Graph visualizing the variance maximization*

Since the variance is defined as the squared euclidean distance to the mean of the data points, here marked in green in the center, projecting the data onto this very vector will yield the highest variance. Similarly, PCA tries to achieve the same for data sets of arbitrary dimensions. Mathematically, we want to find a unit vector u, so that we maximize the variance: $max_{u_1}\sigma_u^2 = \frac{1}{M}\Sigma_{i=1}^{M}(u_1^T x_i - 0)(u_1^T x_i - 0)^T = \frac{1}{M}\Sigma_{i=1}^{M}(u_1^T x_i)(u_1^T x_i)^T = \frac{1}{M}\Sigma_{i=1}^{M}u_1^T(x_i x_i^T)u_1 = u_1^T(\frac{1}{M}\Sigma_{i=1}^{M}x_i x_i^T)u_1 = u_1^T C u_1$. $C = \frac{1}{M}\Sigma_{i=1}^{M}x_i x_i^T$ is defined as the co-variance matrix. After multiplying with u_1 from the left, the following equation results: $u_1\sigma_u^2 = u_1^T C u_1 = C u_1$. Here, $u_1 u_1^T$ is the norm of the unit vector and therefore equal to one. Since the variance is a scalar, this is equal to $\sigma_u^2 u_1 = C u_1$. This definition corresponds to the definition of eigen-values and eigen-vectors and since we want to maximize the term, u_1 is represented by the biggest eigen-vector, eigen-value combination. u_1, the first eigen-vector of the co-variance matrix, also called the first principal component, therefore gives the direction which yields the highest variance. However, we want to map the data into R^d, which means we need to find a vector u_2 to maximize the variance given u_1. In fact, these vectors have to be orthogonal to each other, which means we have to calculate $max_{u_2:u_2 \perp u1}\sigma_{u_2}^2 = u_2^T C u_2$ to derive the second principal component. This term corresponds to the second biggest eigen-vector, eigen-value combination. To derive the missing vectors for $U = [u_1, u_2, ..., u_n]$, we simply use other eigen-value, eigen-vector combinations, sorted by the eigen-values in descending order.

Using this knowledge, the PCA algorithm is fairly simple. First the data set needs to be centralized, which means first the mean of the data is calculated: $\mu = \frac{1}{M}\Sigma_{i=1}^{M}x_i$ and then every data point i in the data set is moved: $z_i = x_i - \mu$. Afterwards the principal components are computed. With regards to the formulas above it is only necesarry to compute the co-variance matrix $C = \frac{1}{M}\Sigma_{i=1}^{M}x_i x_i^T$ and retrieve the corresponding eigen-values and eigen-vectors $\{(\lambda_i, u_i)\}_{i=1}^{n}$, where the λ_i's are sorted in descending order. Afterwards, we choose the dimensionality d and pick the first d eigen-vectors to combine in the matrix $U = [u_1, u_2, ..., u_n]$. Then we compute the linear transformation function for each z_i: $z_i' = U^T * z_i$ and as a result reduce the dimensionality of the data. In this exercise the dimensionality is given with $d = 30$, this means using PCA we calculate the following reduction: $f : R^{1024} \rightarrow R^{30}$. Normally, when choosing the desired dimensionality, one can use the following formula to determine how much variance should be preserved: $p_d = \frac{\Sigma_{i=1}^{d}\lambda_i}{\Sigma_{i=1}^{n}\lambda_i}$. This formula returns a value $\in [0, 1]$ and describes how much variance of the original data still remains when reducing the dimensionality to d. Besides, to show the data in a $2D$ plane, $t-SNE$, an algorithm for dimensionality reduction which is particularly useful to map data into the $2D$ or $3D$ space is used. As input for the $t-SNE$ algorithm the output of the PCA algorithm is utilized, however, since we use built-in functions for that particular computation we will not go into further detail on how $t-SNE$ works.

Results

In the following section, we show the results for performing the PCA algorithm on the given data set. First, a plot concerning the eigen-values at the corresponding eigen-values indices is shown, then a table concerning the variance for different dimensions. Finally, a $2D$ plot depicting the reduced data which was created with the help of $t-SNE$ is integrated into the section.

0.1. Eigen-values

In this subsection, a graph with the eigen-values at their corresponding eigen-values indices is shown. **Figure 2** depicts the sorted eigen-values descending by value at the y-axis, and the index at which the eigen-value is at the x-axis. The function follows the form of a broken rational function, more specifically a scaled $\frac{1}{x}$-function. The values decrease very fast, starting at around 18 for indices smaller than 50, with a small decrease and values close to zero afterwards.

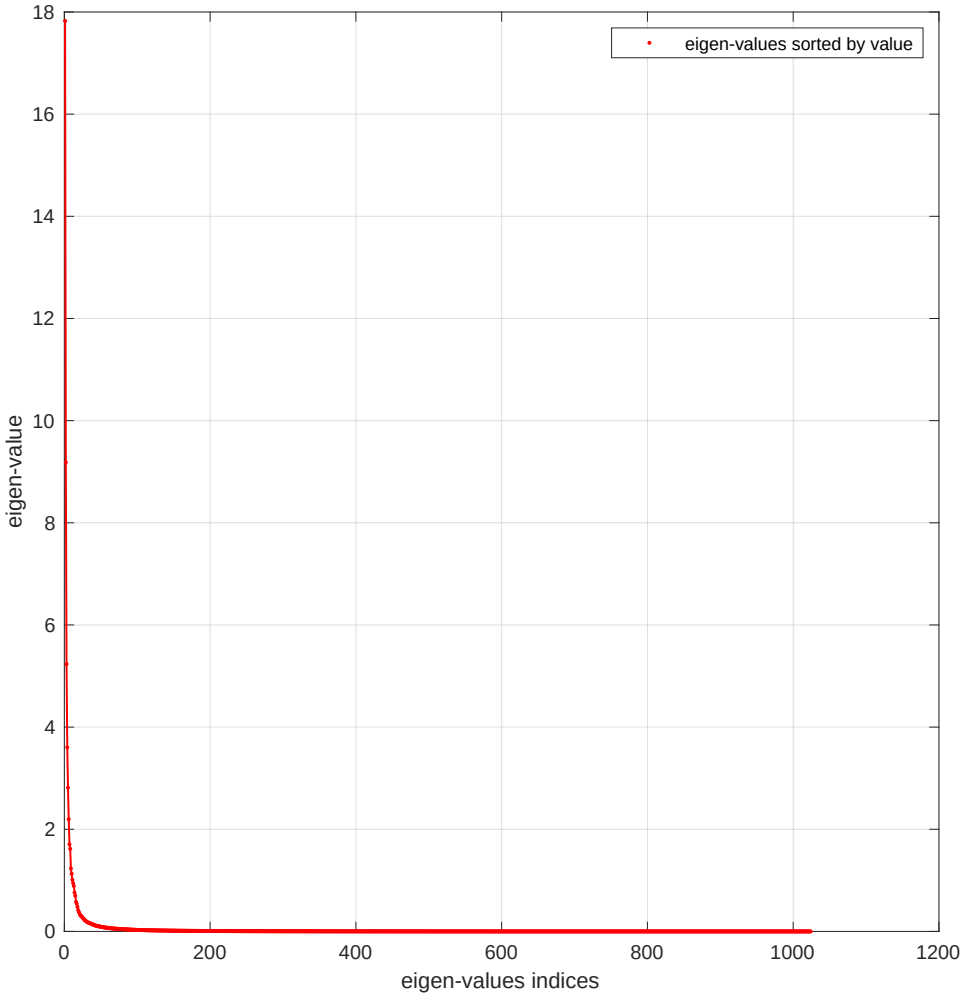

Figure 2: *Graph visualizing the eigen-values at the corresponding eigen-value index*

0.2. Dimensions and variance

This section is concerned with the needed dimensionality d to reach a certain value of variance. **Table 1** shows that even though the increase in variance between the measurements is small, in steps of $0.05 - 0.03$, the needed dimensionality roughly doubles for each increase, starting at $d = 40$ for $\sigma^2 = 0.9$ and reaching $d = 175$ for $\sigma^2 = 0.98$.

variance	dimensionality
0.9	40
0.95	84
0.98	175

Table 1: *Table of the needed dimensionality d for given variance values*

0.3. t-SNE result

The graph in this section, **Figure 3** shows a representation of the data in a $2D$ plane. This representation is derived by applying the $t-SNE$ algorithm on the output of the PCA algorithm, where the data vectors were already dimensionality reduced to $d = 30$. You can see several clusters which are clearly distinct, forming circular groups and one structure in the middle, where the differentiation is not clearly given in a $2D$ plane. Due to a probabilistic approach, the result of $t-SNE$ looks different for every run, but the general structure is preserved.

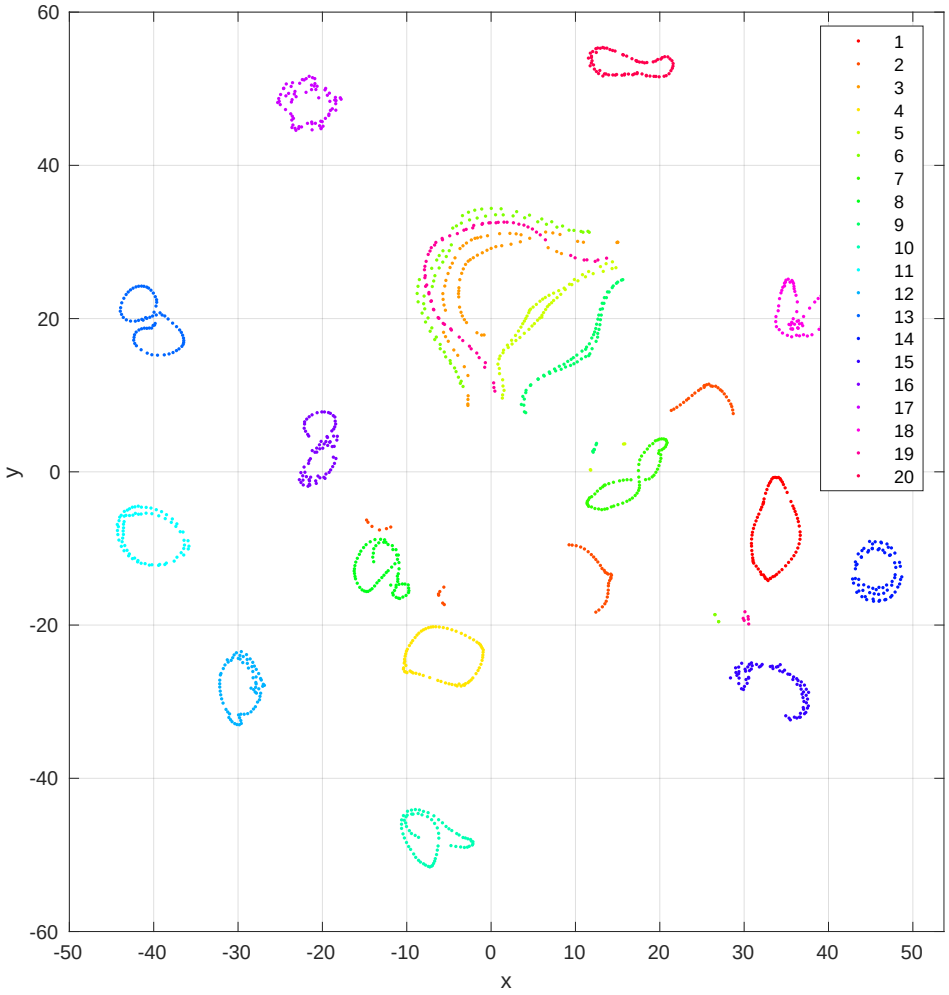

Figure 3: *Graph visualizing the $t-SNE$ results*

Discussion

As described before, the graph giving insight into the eigen-vector, eigen-value connection follows a curve similar to a scaled $\frac{1}{x}$-function. This means that with increasing dimensionality, the newly gained increase in variance reduces. Especially for higher indices, where the eigen-values are close to zero, the maintained variance from the original mapping is insignificant. This observation fits

the data derived from **Table 1**, where for increasing variance thresholds the needed dimensionality increases drastically. Besides, the notice from **Table 1** can also be explained when looking at the formula for deriving the dimensionality directly: $p_d = \frac{\Sigma_{i=1}^{d} \lambda_i}{\Sigma_{i=1}^{n} \lambda_i}$. Since for increasing d, the eigen-values decrease as shown in **Figure 2**, more eigen-values are needed to increase the proportion by the same amount as for lower indices, where the eigen-values are higher in value.

As for the before mentioned difference in the representation of the $t-SNE$ results for various runs, they are a result of the stochastic approach used in the algorithm. In higher dimensions, $t-SNE$ uses Gaussian distribution and when mapping to lower dimension t-distribution is used. This leads to unpredictable results. However, since the big structures are mostly the same for different runs, the data seems to follow a distinct clustering in the higher dimensions. Also, the fact that there is an ambiguous structure in the centre of **Figure 3**, where the assignment of labels to the data is not particularly clear has no expressiveness in lower dimensions because, with the transformation to lower dimensions, information can get lost.

Workload

The workload was split equally, with Lucas Pereira and Matteo Wohlrapp working both on code and report design. Issues and problems were solved in a collaborative manner, with both team partners contributing evenly.