DD2343 Assignment 1

jacmalm@kth.se

November 18, 2021

1 Principal Component Analysis

1.1 Explain why data-centering is required before performing PCA. What might happen if we perform PCA on non-centered data?

PCA attempts to find new axes that are linear combinations of the old axes, such that these new axes, or principal components, minimize the mean squared distance between the original data points and their projection onto the principal axes [3]. One key thing to note is that these principal components are limited by the need to cross through the origin. This limitation means that data that is shifted cannot be approximated as well. A simple example is to consider the points $\{-1, 1\}$, $\{0, 0, \}$, $\{1, -1\}$. These points all lie perfectly along a 1 dimensional manifold, the line with slope -1 going through the origin. The mean of this dataset is 0. Thus, we expect the first principle component to consist of this line, and we expect all of the variance to be explained by the first principle component, as the PCA model is perfectly respected.When we run the PCA code (with the lines responsible for centering the data commented out in the SKLearn library file) given in appendix 1, this is exactly what we see. Great! PCA works.

However, if we shift the location of this manifold, while keeping its shape intact, this will change. If we now consider the points $\{49, 51\}$, $\{50, 50\}$, $\{51, 49\}$, it is easy to see that they also perfectly lie along a line, indeed a line with the same slope as the previous example. However, this line does not pass through the origin, and will thus not be found by PCA. Instead, what we expect to happen, is that the first principle component will be a line that goes through the origin and minimizes the distance from the data points and their projection onto the first principle axis. As the data is symmetric around their mean, we expect this to be a line going through the mean point,$\{50, 50\}$, and the origin.This is a line with slope 1. In fact, this line is perpendicular to the line that the data points lie along. When looking at the explained variance, we can also see that this line does not explain all of the variance, necessitating us to use both principal components to perfectly be able to get back all of our data, meaning that PCA fails to detect the inherent 1-dimensionality of the dataset. Again, these conclusions are verifiable with provided code in Appendix 1, and visually demonstrated in figure 1 and figure 2.

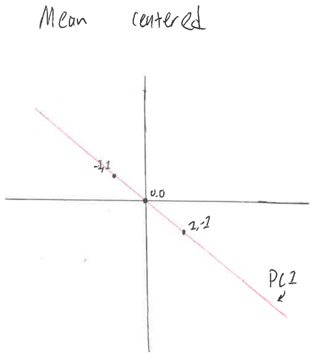


Figure 1.

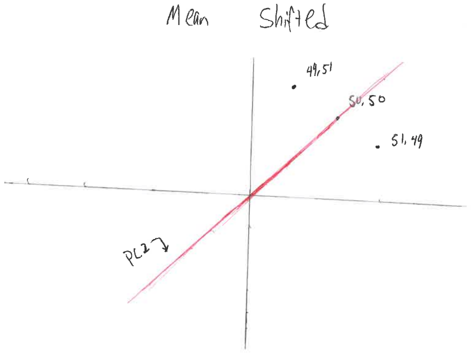


Figure 2.

1.2 Is a single SVD operation sufficient to performing PCA on both the rows and columns of a data matrix

Given a mean-centered data matrix **X** with m columns and n rows, $m < n$, we know that we can estimate the covariance matrix C_X in the following manner:

$$C_X = \frac{1}{n} X X^T$$

We know that this covariance matrix will be symmetric, meaning that it has a spectral decomposition where the eigenvectors are orthogonal, resulting in:

$$C_X = Q \Lambda Q^T$$

This decomposition can be interpreted as capturing the patterns of variance across the dataset **X**. The columns in **Q** provide directions, and their corresponding eigenvalues (as a proportion of the sum of all eigenvalues) encode how much of the variance is captured in the direction of the eigenvector. Thus, the eigenvectors in this decomposition are the principal axes we use to translate our datapoints into our PCA projection [1] .

If we rewrite X in terms of its SVD before computing the covariance matrix, we see that it can be written as

$$C_X = \frac{1}{n} U \Sigma V^T V \Sigma^T U^T$$

Since **U** and **V** are orthonormal, and the only non-zero points of Σ lie along its first diagonal, this expression cancels out to

$$C_X = \frac{1}{n} U \widehat{\Sigma}^2 U^T$$

Allowing us to draw the following conclusions

$$Q = U$$

$$\Lambda = \frac{1}{n} \widehat{\Sigma}^2$$

Where $\widehat{\Sigma}$ is a square, $n \times n$ matrix, where there are m non-zero values all placed along the first diagonal.

That is, the principle axes are given by $\mathbf{V}$ when the corresponding singular values are sorted in order of magnitude.

If we instead want to find the principle axes of the transpose of X, this turns into

$$C_{X^T} = \frac{1}{n} V \Sigma^T U^T U \Sigma V^T$$

$$C_{X^T} = \frac{1}{n} V \overline{\Sigma}^2 V^T$$

Where $\overline{\Sigma}$ is a square, $m \times m$ diagonal matrix.

If we restrict ourselves to looking at non-zero values (the first m values along the first diagonal) $\widehat{\Sigma} = \overline{\Sigma}$, these are the only values that will affect our principle axes.

Thus we can conclude that the principle axes that define our PCA projection of the transpose of X are given by V. This is the same V we get from performing a SVD on X, which means that we get all the necessary information for performing PCA on X and X^T from one SVD operation. We will need to pad/remove zeroes from Σ in order to make the matrix multiplication possible, and we will need to transpose one of the terms we get from the first SVD operation, but we only need to perform it once.

A program which verifies that the principal components of X can be retrieved from an SVD of X^T is given in Appendix 2.

1.3 Explain why the use of the pseudo-inverse is a good choice to obtain the inverse mapping of the linear map

The pseudo-inverse is a good choice to obtain the inverse mapping because when the PCA model is fully respected, that is our data points $\mathbf{y} \in \mathbb{R}^d$ are fully contained within a subspace $\mathbf{U} \in \mathbb{R}^k$ where $k < d$, and $\mathbf{U}$ is the column space of $\mathbf{W}$, $\mathbf{W}^+$ is the inverse mapping of $\mathbf{W}$, even when $\mathbf{W}$ is non-square.

However, when the PCA model is not fully respected and there are points in $\mathbf{y}$ that lie outside of $\mathbf{U}$, the pseudo-inverse computes the inverse mapping from the points projected onto $\mathbf{U}$, which is the approximation that minimizes the euclidean distance between computed and actual points [4].

Thus, using the pseudo-inverse of the linear transformation that is hypothesized to have generated **y** from **x**, allows us to compute the projection onto the PCA axes even for points that do not completely lie within the k dimensional column space, which often occurs due to noisy measurements or an underlying model not quite aligned with the PCA assumptions.

1.4 Derive PCA using the criterion of variance maximization and show that you get the same results as with the criterion of minimizing the reconstruction error

We begin by assuming that our data set are observations from a set of d jointly distributed gaussian variables, $\mathbf{Y}$, that have undergone a linear transformation $\mathbf{W}$ applied to a set of p latent variables $\mathbf{X}$, where $p < d$. This linear transformation is restricted to be an axis change, meaning that $\mathbf{W}$ is orthogonal.

We assume that the latent variables $\mathbf{x}$ are uncorrelated. This means that the covariances are zero, and that the covariance matrix C_x is diagonal.

The covariance of $\mathbf{x}$ is given by

$$C_x = \mathbb{E}[xx^T]$$

Similarly, the covariance of y is given by

$$C_y = \mathbb{E}[yy^T]$$

Rewriting this equation using the PCA assumptions ($y = Wx$)

$$C_y = \mathbb{E}[Wxx^TW^T]$$

Using the fact that expectation is a linear operator

$$C_y = W\mathbb{E}[xx^T]W^T$$

$$C_y = WC_xW^T$$

Since **W** is orthonormal

$$W^TC_y = W^TWC_xW^T$$

$$W^TC_yW = W^TWC_xW^TW$$

$$W^TC_yW = IC_xI$$

$$W^TC_yW = C_x$$

Since C_y is symmetrical we can use its spectral decomposition to rewrite it as

$$C_y = Q\Lambda Q^T$$

And thus

$$C_x = W^TQ\Lambda Q^TW$$

Remembering our assumption that the variables in **x** are uncorrelated, C_x is a diagonal matrix. Since Λ is diagonal and Q is orthogonal,

$$WQ^T = I$$

$$WQ^TQ = QI$$
$$W = Q$$

and

$$C_x = \Lambda$$

However, since C_X is a $p \times p$ matrix only the first p eigenvalues of Λ are non-zero (assuming the PCA model is followed). Thus the above equality only holds when choosing the p first eigenvectors when sorting by corresponding eigenvalue size, and restricting Λ to its non-zero diagonal values.

This suggests that the eigenvalues of the spectral decomposition of C_y represent the amount of variance captured in the direction of its corresponding eigenvector, with our eigenvectors spanning the linear subspace that contains $\mathbf{y}$. Thus our principal axes are given by Q in order of importance, when Λ is sorted in descending order [2].

Similarly, when deriving PCA from a minimization of squared error perspective, our principal components are given by $\mathbf{U}$, with $X = U\Sigma V^T$. As shown in 1.2, $U = Q$, and Σ^2 is proportional to Λ. Thus, due to the monotonicity of the square function, the principal axes will appear in the same order, and have the same values. As our principal axes determine our PCA projection, we can conclude that the two derivations yield equivalent results [2].

2 Multidimensional Scaling and Isomap

2.1 Explain intuitively why the double centering trick is needed to be able to solve for S given D

Given a squared distance matrix **D** we wish to find Gram matrix **S**. As mentioned, the equation for this conversion is given by

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$$

The issue here is that we do not know the coordinates of our data points and as a result s_{ii}, s_{jj} are unknown, making this definition circular. When we assume that our data is centered (which we are allowed to do since distance is invariant to location), it can be shown that

$$-(s_{ii} + s_{jj}) = \mu_{ij} - \mu_i - \mu_j$$

These μ's refer to the means of the distance matrix we are given. Thus, by assuming that our data is centered, we can rewrite the unknowns in our conversion equation in terms of information contained in the distance matrix that we are given. This makes the conversion possible [2].

The double centering trick simply performs

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 + (\mu_{ij} - \mu_i - \mu_j))$$

for all matrix elements in **D**, converting our given distance matrix into the desired Gram matrix.

2.2 Use the same reasoning as in the previous question to argue that s_{ij} can be computed as $s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$

Comparing the original equation

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$$

with

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$$

We can deduce that

$$s_{ii} + s_{jj} = d_{1i}^2 + d_{1j}^2$$

must hold.

From Lee and Verleysen [2] equation 4.42 we know that we can rewrite this equation accordingly

$$s_{ii} + s_{jj} = (\langle \mathbf{y_1} \cdot \mathbf{y_1} \rangle - 2\langle \mathbf{y_1} \cdot \mathbf{y_i} \rangle + \langle \mathbf{y_i} \cdot \mathbf{y_i} \rangle) + (\langle \mathbf{y_1} \cdot \mathbf{y_1} \rangle - 2\langle \mathbf{y_1} \cdot \mathbf{y_j} \rangle + \langle \mathbf{y_j} \cdot \mathbf{y_j} \rangle)$$

Where $\mathbf{y_1}$ represents the vector holding the coordinates of the first point. Now, if instead of assuming that our data points are mean centered and assume that the first data point is located on the origin, which we are allowed to do since this does not affect the distance between any of the points, the above equation reduces to

$$s_{ii} + s_{jj} = \langle \mathbf{y_j} \cdot \mathbf{y_j} \rangle + \langle \mathbf{y_i} \cdot \mathbf{y_i} \rangle$$

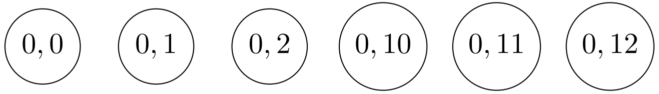
$$s_{ii} + s_{jj} = s_{ii} + s_{jj}$$

Thus we can see that we are justified in using this alternate equation to calculate our Gram matrix. It will not result in the same mapping as if we used $s_{ij} = -\frac{1}{2}(d_{ij}^2 + (\mu_{ij} - \mu_i - \mu_j))$, which will be mean centered, whereas our equation will fix the location of the first data point to the origin.

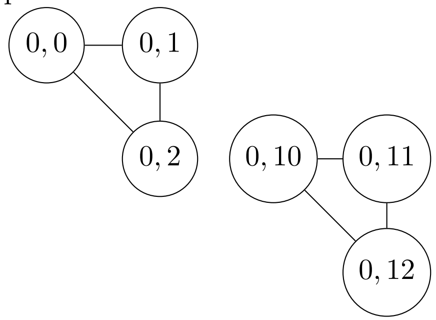
2.3 Argue that the process to obtain the neighbourhood graph G in the Isomap method may yield a disconnected graph. Provide an example. Explain why this is problematic.

The Isomap algorithm works similarly to MDS in that it relies on computing a Gram matrix **S** from a distance matrix **D**. However, dissimilarly to MDS, this distance matrix is not made up of euclidean distances. Instead, the distances are approximations of the shortest path from point A to point B where the path is restricted to the theoretical manifold our data distribution makes up. Computing the shape of this manifold as well as the length of the path through the manifold from point A to B is infeasible. An approximation of the manifold can be created by drawing a neighbourhood graph G. This neighbourhood graph G is created one of two ways. Either by joining each point to its k nearest neighbours, where a point A is considered closer to point B than to C if the euclidean distance between A-B is smaller than A-C. Alternatively the neighbourhood graph can be constructed by choosing to join a point A to all of its neighbours that lie within a certain distance d.

To demonstrate how this process can lead to a disconnected graph, assume we have the following nodes located at given coordinates.



If we go by the k-nearest neighbour approach and set k=2, or go by max euclidean distance between nodes and set d to smaller than 8, our resulting graph will be disconnected.



According to the Isomap algorithm, when we compute the distance matrix, the entry between point A and point B will be the summation of the distances between all nodes that lie along the shortest path between A and B through the neighbourhood graph G constructed in the previous step. If this is a disconnected graph, the distance between any point A to any point B where A and B are members of different disjoint subsets in G will be infinite, or undefined. In either of these cases the spectral decomposition of the Gram matrix corresponding to the distance matrix will be undefined and the Isomap algorithm will fail.

References

[1] *Covariance and Principal Component Analysis (PCA).*

[2] *Nonlinear Dimensionality Reduction.* Springer, 2007.

[3] *Advanced Data Analysis from an Elementary Point of View.* Cambridge University Press, 2013.

[4] Guido Gerig. Least squares, pseudo-inverses, pca and svd. Technical report, University of Utah, 2012.

3 Appendices

3.1 Appendix 1

```python
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[49, 51], [50, 50], [51, 49]])
X_mean_centered = np.array([[-1, 1], [0, 0], [1, -1]])
pca = PCA()
pca.fit(X)
print(pca.components_)
print(pca.singular_values_)
print(pca.explained_variance_ratio_)
```

```
pca.fit(X_mean_centered)
print(pca.components_)
print(pca.singular_values_)
print(pca.explained_variance_ratio_)
```

3.2 Appendix 2

```
from sklearn.decomposition import PCA
import numpy as np

X = np.array([[1, 2], [3, 4], [5, 6]], dtype=float)
X -= np.mean(X, axis=0)
Xt = np.matrix.transpose(X)

pca = PCA()
pca.fit(X)

# now our goal is to recreate these principal components using SVD of transposed data
V, Sigma_t, U_t = np.linalg.svd(Xt)
#change the params since we want to get transposed SVD
V_t = np.matrix.transpose(V)

for i in range(pca.singular_values_.size):
    print(pca.singular_values_[i], pca.components_[i])

for i in range(len(Sigma_t)):
```

```
print(Sigma_t[i], V_t[i])
```