MDS PCA Equivivalence

Carl Henrik Ek che29@cam.ac.uk

November 24, 2021

Abstract

This document will outline the derivation of Multi-dimensional scaling (MDS) and show the equivivalence between MDS and Principal Component Analysis (PCA). We will derive PCA from a probabilistic model that leads to an algorithm that is the same as MDS. We will show that even though similar they have clearly different benefits in how they can be non-linearised. We will show how PCA leads to the GP-LVM and how MDS leads to a whole set of different methods known as spectral dimensionality reduction methods.

Introduction

Before we start there are a couple of mathematical things that we need to know about. In specific we are going to use eigen-decompositions a lot. The eigen decomposition is a similarity transform and it can be worthwhile seeing the more general definition of this before we move on.

Lets assume that we have two linear maps $\mathbf{M}_A$ and $\mathbf{M}_B$ and a *change-of-basis* transformation $\mathbf{P}$ that maps from basis A to B. This means that we have the following relationship between the different maps,

$$\begin{tikzcd}
\mathbf{x}_A \arrow[r, "\mathbf{M}_A"] \arrow[d, "\mathbf{P}"'] & \mathbf{M}_A\mathbf{x}_A \\
\mathbf{x}_B \arrow[r, "\mathbf{M}_B"'] & \mathbf{M}_B\mathbf{x}_B \arrow[u, "\mathbf{P}^{-1}"']
\end{tikzcd} \tag{1}$$

Now we can write up the relationship between the two linear maps,

$$\mathbf{M}_B\mathbf{x}_B = \mathbf{P}\left(\mathbf{M}_A\mathbf{x}_A\right) = \mathbf{P}\left(\mathbf{M}_A\left(\mathbf{P}^{-1}\mathbf{x}_B\right)\right) = \left(\mathbf{P}\mathbf{M}_A\mathbf{P}^{-1}\right)\mathbf{x}_B$$

This means that we have the following relationship between the two linear maps,

$$\mathbf{M}_B = \mathbf{P}\mathbf{M}_A\mathbf{P}^{-1},$$

which we refer to as saying that the maps are *similar* or $\mathbf{M}_A \sim \mathbf{M}_B$.

Similar matrices have several properties that are quite useful for the methods that we will look at specifically these things will become useful. If $\mathbf{A} \sim \mathbf{B}$ then,

$$\det(\mathbf{A}) = \det(\mathbf{B}) \tag{2}$$
$$\text{trace}(\mathbf{A}) = \text{trace}(\mathbf{B}) \tag{3}$$
$$\mathbf{A}^m \sim \mathbf{B}^m, \quad m \in \mathbb{Z}^+ \tag{4}$$
$$\mathbf{A} \text{ invertible if and only if } \mathbf{B} \text{ invertible}, \tag{5}$$

in other words the determinant, trace and invertability is invariant under similarity.

Now the most commonly used similarity transform is the spectral decomposition where one of the matrices is diagonal,

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} \tag{6}$$

$$\mathbf{\Lambda} = \begin{cases} 0 & i \neq j \\ \lambda_i & i = j \end{cases} \tag{7}$$

$$\mathbf{V}\mathbf{V}^{\mathrm{T}} = \mathbf{I} \quad \Rightarrow \quad \mathbf{V}^{-1} = \mathbf{V}^{\mathrm{T}}. \tag{8}$$

One of the nice things with the eigendecomposition is that it implies a really simple way of writing any square matrix as a sum of rank 1 matrices,

$$\mathbf{M} = \sum_{i=1}^{N} \lambda_i \mathbf{v}_i \mathbf{v}_i^{\mathrm{T}}.$$

There are many useful things to represent a matrix using its diagonal form especially considering the invariances that we stated above. The spectral theorem states that all symmetric real matrices have a diagonal matrix they are similar to. This is normally just stated as a fact but its quite interesting to actually show this. I'm not going to do this here but if you are interested have a look at this PDF which does a good job of walking through this.

Rank-Nullity Theorem

The next thing we need to go through has to do with the objective of what we want to do. In most cases we are interested in finding a reduced dimensionality of the data. We will refer to two different representations of the data, the original representation of the data in its raw form where the dimensionality we talk about is the degrees of freedom of the representation. The second representation is the cannonical representation of the *data* this the number of degrees of freedom in the data. An example would be a line in a two dimensional representation, where the cannonical representation is one or in a non-linear case the classic swiss-roll data which is two-dimensional but embedded in a representation of three. The relationship between these two concepts is clearly stated through the *rank-nullity theorem.*

The theorem states this, given two vector spaces A and B and a transformation $T : A \rightarrow B$,

$$\mathrm{Rank}(T) + \mathrm{Nullity}(T) = \dim(A)$$

where $\mathrm{Rank}(T)$ is the dimensionality of the *image* of T and $\mathrm{Nullity}(T)$ is the dimensionality of the *kernel* of T[1]. So with linear maps this becomes quite simple, say that we have a map $\mathbf{M} \in \mathbb{R}^{m \times n}$ now this will lead to the following relationship[2],

$$\underbrace{\mathrm{Rank}(\mathbf{M})}_{\min(m,n)} + \underbrace{\mathrm{Nullity}(\mathbf{M})}_{\max(0,n-m)} = n.$$

Now the task of any dimensionality reduction method is to find a map, explicit or implicit, that makes the *nullity* of the transformation to be the domain where the data have no variations this means that the *image* of the transformation is the canonical representation of the data. In the world of noise and applications we are likely to have a scenario where we want to remove [3] variations from the original representation that we do not want represented in our embedding. Doing so differentiates maths from machine learning as this implies that we need to make some assumptions of what we want to remove and this is where it starts getting very interesting as this requires models.

[1]Where the kernel is the set of all points that maps to zero i.e. $\mathrm{kernel}(T) = \{\mathbf{x} : \mathbf{x} \in A \mid T(\mathbf{x}) = 0\}$
[2]If we assume that the transformation T is "full rank" either in its column or row space.
[3]Or in a Bayesian setting "explain away"

Inner-products to Representations

There is two more things that we will need in before we can start looking at the methods and that is one how we can convert between a distance matrix and an inner-product matrix and how we can take an inner-product matrix and provide a possible vector representation that could have generated the matrix.

We will refer to the euclidean distance matrix as $\mathbf{D}$ where,

$$\mathbf{D}_{ij}^2 = d_{ij}^2 = \sum_{k=1}^{d}(y_{ki} - y_{kj})^2 = \mathbf{y}_i^{\mathrm{T}}\mathbf{y}_i + \mathbf{y}_j^{\mathrm{T}}\mathbf{y}_j - 2\mathbf{y}_i^{\mathrm{T}}\mathbf{y}_j,$$

where d is the dimensionality of the representation. We will then refer to the inner-product matrix or the Gram matrix $\mathbf{G}$ as,

$$\mathbf{G}_{ij} = g_{ij} = \sum_{k=1}^{d} y_{ki}y_{kj} = \mathbf{y}_i^{\mathrm{T}}\mathbf{y}_j.$$

This leads to the simple relationship that,

$$d_{ij}^2 = g_{ii} + g_{jj} - 2g_{ij}.$$

In the algorithmic part we will in general only be able to determine the representation up to an affine transform. To fix one degree of freedom we will enforce the constraint that the data is centered. For the gram matrix this means,

$$\sum_{i=1}^{N} g_{ij} = \sum_{i=1}^{N}\mathbf{y}_i^{\mathrm{T}}\mathbf{y}_j = \underbrace{\left(\sum_{i=1}^{N}\mathbf{y}_i^{\mathrm{T}}\right)}_{\mathbf{0}}\mathbf{y}_j = 0.$$

Now our task is to express a general element in the gram matrix g_{ij} completely in terms of distances. We can do this by first noting that,

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

this means that we only need to find a representation for the diagonal elements of the gram matrix to "correct" the distance. We will do this by using the centring constraint that we specified before which will allow us to isolate the diagonal elements. We will start by isolating a general diagonal term,

$$g_{ij} = \frac{1}{2}(g_{ii} + g_{jj} - d_{ij}^2) \tag{9}$$

$$\sum_{i=1}^{N} g_{ij} = \sum_{i=1}^{N}\frac{1}{2}(g_{ii} + g_{jj} - d_{ij}^2) \tag{10}$$

$$0 = \frac{1}{2}\left(Ng_{jj} + \sum_{i=1}^{N} g_{ii} - \sum_{i=1}^{N} d_{ij}^2\right) \tag{11}$$

$$\sum_{i=1}^{N} d_{ij}^2 = \sum_{i=1}^{N} g_{ii} + Ng_{jj} \tag{12}$$

$$g_{jj} = \frac{1}{N}\left(\sum_{i=1}^{N} d_{ij}^2 - \sum_{i=1}^{N} g_{ii}\right). \tag{13}$$

We will now re-write the trace of the gram matrix using the distance matrix by adding a sum over the remaining index in in Eq.12,

$$\sum_{i=1}^{N} d_{ij}^2 = \sum_{i=1}^{N} g_{ii} + N g_{jj} \tag{14}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} g_{ii} + \sum_{j=1}^{N} N g_{jj} \tag{15}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 = N\sum_{i=1}^{N} g_{ii} + N\sum_{i=1}^{N} g_{jj} \tag{16}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 = 2N\sum_{i=1}^{N} g_{ii} \tag{17}$$

$$\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 = \sum_{i=1}^{N} g_{ii} = \mathrm{trace}(\mathbf{G}). \tag{18}$$

Now we have everything that we need to go back and write the general element of the gram matrix by Eq. inserting Eq.13 and 18 into Eq. 9 which leads to the following,

$$g_{ij} = \frac{1}{2}\left(\frac{1}{N}\left(\sum_{k=1}^{N} d_{kj}^2 - \frac{1}{2N}\sum_{k=1}^{N}\sum_{p=1}^{N} d_{kp}^2\right) + \frac{1}{N}\left(\sum_{k=1}^{N} d_{ik}^2 - \frac{1}{2N}\sum_{k=1}^{N}\sum_{p=1}^{N} d_{kp}^2\right) - d_{ij}^2\right) \tag{19}$$

$$= \frac{1}{2}\left(\frac{1}{N}\left(\sum_{k=1}\left(d_{kj}^2 + d_{ik}^2\right) - \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{N} d_{kp}^2\right) - d_{ij}^2\right) \tag{20}$$

There are many different (and much nicer possibly) ways of deriving this and you can write it much nicer on matrix form using **1** matrices to deal with the sums that appears but hopefully this one makes sense as well as it was what came most natural for me.

Now that we can convert between a distance matrix to a gram matrix we can easily find a possible vectorial representation that could have generated this matrix by,

$$\mathbf{G} = \mathbf{Y}\mathbf{Y}^{\mathrm{T}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} \tag{21}$$

$$= \left(\mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}\right)\left(\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{V}^{\mathrm{T}}\right) \tag{22}$$

$$= \left(\mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}\right)\left(\mathbf{V}\left(\mathbf{\Lambda}^{\frac{1}{2}}\right)^{\mathrm{T}}\right)^{\mathrm{T}} \tag{23}$$

$$= \left(\mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}\right)\left(\mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}\right)^{\mathrm{T}}, \tag{24}$$

which means we can simply choose our representation that have generated the Gram matrix to be,

$$\mathbf{Y} = \mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}.$$

Multi Dimensional Scaling

Now when we have set-up the background it is time to have a look at the algorithm that underpins most of the classical dimensionality reduction methods namely Multi-Dimensional-Scaling (MDS) Cox et al., 2008. The idea is simple, we are given a *complete* set of similarity or proximity measures between a finite set of "entities". Now the task is to find a geometric configuration of these entities that preserves the proximity relationship. In other words, we want the euclidean distance to "encode" the given similarity.

Given N entities and a matrix of proximity relationships between them $\mathbf{\Delta}$ where $\mathbf{\Delta}_{ij}$ is the relationship between entity i and j. The task is now to find an embedding $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$ such that $\|\mathbf{y}_i - \mathbf{y}_j\|_{L2} \approx \delta_{ij}$. In MDS this is formulated as minimising the frobenious norm between the given proximity matrix $\mathbf{\Delta}$ and the one calculated directly from the embedding $\mathbf{D}$ where $\mathbf{D}_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|_{L2}$.

$$\hat{\mathbf{Y}} = \text{argmin}_{\mathbf{Y}} \|\mathbf{D} - \mathbf{\Delta}\|_F. \tag{25}$$

Now we will make use of two rules for similar matrices and in specific the eigendecomposition to decompose the matrices into diagonal. First, the Frobenius norm is very simple, in effect it is just the generalisation of the euclidean norm to matrices. The general element-wise norm for matrices $L_{p,q}$ norms takes the following form,

$$\|\mathbf{M}\|_{p,q} = \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{m} |m_{ij}|^p \right)^{\frac{p}{q}} \right)^{\frac{1}{q}}$$

where the Frobenius norm is the one where $p = q = 2$. Now lets say that we have a square diagonal matrix that we want to compute the Frobenious norm of. In this case the norm is just the square root of the sum of the square of the diagonal elements which can easily be written as,

$$\|\mathbf{M}\|_F = \sqrt{\text{trace}(\mathbf{M}^{\mathrm{T}}\mathbf{M})} = \sqrt{\text{trace}(\mathbf{M}^2)}.$$

Now we will use two rules of similar matrices, we know that if $\mathbf{A} \sim \mathbf{B}$ then $\mathbf{A}^2 \sim \mathbf{B}^2$ and we also know that the trace is invariant under similarity. These two things means that if we diagonalise a matrix through its eigendecomposition the frobenious norm is simply the square root of the sum of the eigenvalues.

Using this we can now come back to the optimisation problem specified by MDS 25,

$$\text{argmin}_{\mathbf{D}} \|\mathbf{D} - \mathbf{\Delta}\|_F^2 = \text{argmin}_{\mathbf{D}} \text{trace} \left(\mathbf{D} - \mathbf{\Delta}\right)^2. \tag{26}$$

Now we will write the matrix Δ using its eigendecomposition and the also re-write the matrix $\mathbf{D}$ using a similar matrix as,

$$\text{argmin}_{\mathbf{D}} \text{trace} \left(\mathbf{D} - \mathbf{\Delta}\right)^2 = \text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}} - \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} \right)^2. \tag{27}$$

Now using the fact that the trace is invariant under similarity transform we can pre and post multiply with the known eigen-vectors of the proximity matrix $\mathbf{V}$,

$$\begin{aligned}
\text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}} - \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} \right)^2 &= \text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{V}^{\mathrm{T}} \left(\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}} - \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}} \right) \mathbf{V} \right)^2 && (28) \\
&= \text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{V}^{\mathrm{T}}\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}}\mathbf{V} - \mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V} \right)^2 && (29) \\
&= \text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{V}^{\mathrm{T}}\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}}\mathbf{V} - \mathbf{\Lambda} \right)^2. && (30)
\end{aligned}$$

Now from the above we can see that we have a diagonal matrix $\mathbf{\Lambda}$ that we want to match by choosing $\mathbf{Q}$ and $\hat{\mathbf{\Lambda}}$. This means that as $\hat{\mathbf{\Lambda}}$ is diagonal the minima will be achieved when $\mathbf{V}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$. This leads to,

$$\begin{aligned}
\text{argmin}_{\mathbf{Q},\hat{\mathbf{\Lambda}}} \text{trace} \left(\mathbf{V}^{\mathrm{T}}\mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{\mathrm{T}}\mathbf{V} - \mathbf{\Lambda} \right)^2 &= \text{argmin}_{\hat{\mathbf{\Lambda}}} \text{trace} \left(\hat{\mathbf{\Lambda}} - \mathbf{\Lambda} \right)^2 && (31) \\
&= \sum_{i=1}^{N} (\hat{\lambda}_i - \lambda_i)^2. && (32)
\end{aligned}$$

To provide a rank d approximation $\mathbf{D}$ to $\mathbf{\Delta}$ we should therefore match the largest d eigenvalues and choose,

$$\mathbf{D} = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_i^{\mathrm{T}}, \tag{33}$$

where the eigenvalues have been sorted such that $\lambda_i \geq \lambda_j$ when $i < j$. This leads to the following error of the embedding,

$$\|\mathbf{D} - \mathbf{\Delta}\|_F = \sqrt{\sum_{i=d+1}^{N} \lambda_i^2}.$$

Now that we have the rank constrained matrix $\mathbf{D}$ that minimises the MDS objective we can convert this to an inner-product matrix and from this pick the representation using what we derived in Section .

Now when we have derived the algorithm its time to reflect a bit about what this method actually does. First it makes a hard assumption that there exists a metric representation that reflects the proximity. This means that the matrix has to be positive semi-definite. The second assumption is less clear, what does minimising the Frobenious norm actually imply? Think about this now, it is not completely obvious to me at this stage but later we will see that by a slightly different derivation we can see what assumption this implies.

Generation of $\mathbf{\Delta}$

The basis of MDS is the proximity matrix $\mathbf{\Delta}$. This matrix can be generated in many different ways. One example I've always thought of is placing people at a dinner table, if I could come up with a number of how much I think people will enjoy to talk to each other and from this I want to generate the actual seating arrangement. Over the years there has been a lot of work of coming up with different ways of computing ad-hoc distances directly from a representation and then apply MDS on this distance. This is the foundation of a lot of different algorithms usually referred to as Manifold Learning.

The idea is rather simple, we have some observed data $\mathbf{Y}$ and we believe that this data lies on a manifold that we want to "unravel". The definition of a manifold is,

Definition 1. *A Manifold is a topological space which locally resembles a Eucledean space for each point.*

This means that locally distances along the manifold are well approximated by the euclidean distance in the observed representation. Now if we would have a some form of neighbourhood structure that told us between which points the euclidean distance is a good approximation of the distance along the manifold we could at least compute a partial proximity matrix Δ.

There are many different approaches to doing just this, the simplest is one called Isomap Tenenbaum et al., 2000 where first a neighborhood graph is constructed from the observed data by making the assumption that say the k closest points to each point obeys the assumption. Then the euclidean distance is computed between these. Now to run MDS we need a complete matrix and in Isomap this is generated by computing the shortest path in the local neighborhood graph and adding up the local distance of this path. Now we have a complete proximity graph to run MDS on. I think this is a really cute and clever approach, hopefully you understood enough from these few lines to code this up in about 10 lines of `Python`.

There are many other methods that basically does the same thing, coming up with different ways of generating the full proximity matrix from a sparse matrix induced by a neighbourhood graph. Of all the methods the one that I think is most interesting is called Maximum Variance Unfolding Weinberger et al., 2004. The interesting thing with this one is that it makes use of the fact that the set of positive definite matrices is convex and it formulates a Semi-Definite-Program that learns the matrix. Neil Lawrence have also written a paper that provides a bit more modelling perspective on this which is an interesting read Lawrence, 2012.

Principal Component Analysis

Principal Component Analysis (PCA) is probably the most important method when it comes to data-science. What is still confusing is that the algorithm that solves it seems to be better known than the model itself. Here we will first go through and derive the model and then from this derive the algorithm that solves it.

PCA was initially proposed by Hotelling Hotelling, 1933 who described a model. However, quite often the reference that is used is instead to an earlier paper by Spearman Spearman, 1904. In this paper a method called Factor Analysis is proposed. However, this method is ill-determined and requires additional constraints to be solved. And as it just happens the one that is normally used is equivalent to PCA. What we will do here is take the slightly more general approach and derive PCA through a full probabilistic model based on Tipping et al., 1999. In Lawrence, 2005 the connection between this derivation of PCA and the non-linear version is described[4].

The aim of unsupervised learning is to observe a set of data $\mathbf{Y}$ and make an assumption that this data have been generated from a latent representation $\mathbf{X}$ through some mapping $f(\cdot)$ such that,

$$\mathbf{y}_i = f(\mathbf{x}_i). \tag{34}$$

Now our task is to recover both the latent representation and the mapping. This seems like a really odd problem at first. A very simple solution to it is to say that the mapping is the identity meaning that the latent and the observed data is the same[5]. So we have to add some form of supervision, some form of direction that tells us what it is that we are looking for. One very general scenario is that we might look for a lower dimensional representation of the data. This is often beneficial as many algorithms scale very badly with dimension so if we can reduce the dimensionality of $\mathbf{Y}$ from say $\mathbb{R}^D$ to $\mathbb{R}^Q$ if $Q < D$ then we can apply our expensive algorithm on the latent representation instead. But there might be lots of other demands as well where for one reason or the other we wish to reorganise our data in a different manner than it was presented.

The important thing is that the problem that we are trying to solve is badly constrained, there are clearly so many possible solutions so that we need to somehow encode our preference towards certain solutions. Now here comes a slightly different interpretations of assumptions or beliefs namely preference. They actually play a very similar role, given two equal solutions under the data we preferred the one we believed in more, now that is just the same as a preference. So another way to think of priors is as preferences over the different hypothesis.

We will first formulate a likelihood of the data. In PCA this is assumed to be a standard Gaussian likelihood,

$$p(\mathbf{Y}|\mathbf{F}) = \mathcal{N}(\mathbf{F}, \beta^{-1}\mathbf{I}). \tag{35}$$

We will start with a model where we use a linear model for our mapping from the latent to the observed space. We will use the same model as linear regression case with the only difference being that as we do not know the input we are going to specify a preference/belief over this. First we have a likelihood function which describes how likely observations is under our model,

$$p(\mathbf{Y}|\mathbf{W}, \mathbf{X}) = \mathcal{N}(\mathbf{X}\mathbf{W} + \mu, \beta^{-1}\mathbf{I}), \tag{36}$$

where μ is a constant offset. We will then use a Gaussian prior over both the weights $\mathbf{W}$ and the latent coordinates $\mathbf{X}$. This leads to the following joint distribution,

$$p(\mathbf{Y}, \mathbf{W}, \mathbf{X}) = p(\mathbf{Y}|\mathbf{W}, \mathbf{X})p(\mathbf{X})p(\mathbf{W}). \tag{37}$$

Our aim is now to derive the posterior distribution over the unknown parameters $\mathbf{W}$ and $\mathbf{X}$. To do so we need to marginalise out these parameters from the joint defined above. This means we need to compute,

$$p(\mathbf{Y}) = \int p(\mathbf{Y}|\mathbf{W}, \mathbf{X})p(\mathbf{X})p(\mathbf{W}). \tag{38}$$

Sadly the following integral is not tractable, we cannot solve it for both $\mathbf{W}$ and $\mathbf{X}$. In order to proceed we are going to integrate out one of the variables and take a point estimate for the other. Now we could either

[4]Now with more powerful inference methods we do not need to think about choosing what to integrate out as is described in this paper, we can approximately do both using variational methods or stochastic inference.

[5]I like to use this analogy. You know that I am here at the moment, now give me a starting location and a path of how I got here. To this question there is infinitely many answers.

choose $\mathbf{W}$ or $\mathbf{X}$ to optimise for so which one should we choose? Well lets say that we have N data points and we are looking for a Q dimensional latent representation. This means that $\mathbf{W} \in \mathbb{R}^{D\times Q}$ and $\mathbf{X}^{N\times Q}$ assuming that $N > D$ it makes sense to treat as many of the random variables with principle as we can and integrate out $\mathbf{X}$ while we optimise $\mathbf{W}$. Another motivation is that if we are given new data and we want to infer its latent location then we want to have the posterior distribution $p(\mathbf{x}|\mathbf{y})$ and we can only reach this distribution if we marginalise out the latent space.

First we are going to make use of the fact that the product of two Gaussians is also a Gaussian. So if we multiply our likelihood and our prior we should now again have a Gaussian, Now we know both of the terms on the left hand-side of the expression,

$$p(\mathbf{X}, \mathbf{Y}|\mathbf{W}) = p(\mathbf{Y}|\mathbf{W}, \mathbf{X})p(\mathbf{X}). \tag{39}$$

$$\begin{aligned} p(\mathbf{Y}|\mathbf{W}, \mathbf{X}) &= \mathcal{N}(\mathbf{X}\mathbf{W} + \mu, \beta^{-1}\mathbf{I}) & (40)\\ p(\mathbf{X}) &= \mathcal{N}(\mathbf{0}, \mathbf{I}). & (41) \end{aligned}$$

We will now derive the joint distribution of a pair of points, $\mathbf{y}$ and $\mathbf{x}$ rather than the full data matrix. As we know that the product is a Gaussian we can write up the definition of the terms in this joint distribution,

$$p(\mathbf{y}, \mathbf{x}|\mathbf{W}) = \mathcal{N}\left(\left[\begin{array}{c} \mathbb{E}[\mathbf{y}] \\ \mathbb{E}[\mathbf{x}] \end{array}\right], \left[\begin{array}{cc} \mathbb{E}[(\mathbf{y} - \mathbb{E}[\mathbf{y}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^{\mathrm{T}}] & \mathbb{E}[(\mathbf{y} - \mathbb{E}[\mathbf{y}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^{\mathrm{T}}] \\ \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^{\mathrm{T}}] & \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^{\mathrm{T}}] \end{array}\right]\right). \tag{42}$$

We will now walk through each of the expectations in turn to specify the joint distribution above. Once we have the joint we can use the by know well know identities to get the marginal and the posterior distribution that we are looking for. Let us start with the means,

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \mathbf{0} & (43)\\ \mathbb{E}[\mathbf{y}] &= \mathbb{E}[\mathbf{W}\mathbf{x} + \mu + \epsilon] = \mathbb{E}[\mathbf{W}\mathbf{x}] + \mathbb{E}[\mu] + \mathbb{E}[\epsilon] & (44)\\ &= \mathbf{W}\mathbb{E}[\mathbf{x}] + \mathbb{E}[\mu] + \mathbb{E}[\epsilon] = \mathbf{0}\mathbf{W} + \mu + \mathbf{0} & (45)\\ &= \mu & (46) \end{aligned}$$

The mean of the latent variable is easy as it come directly through its definition. The one for the observed data comes through the model of the observed data so we first need to rewrite it in terms of the model. Now when we have the means we can move on to deriving the elements of the covariance matrix,

$$\begin{aligned} \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^{\mathrm{T}}] &= \mathbb{E}[(\mathbf{x} - 0)(\mathbf{y} - \mu)^{\mathrm{T}}] & (47)\\ &= \mathbb{E}[\mathbf{x}(\mathbf{W}\mathbf{x} + \mu + \epsilon - \mu)^{\mathrm{T}}] & (48)\\ &= \mathbb{E}[\mathbf{x}(\mathbf{W}\mathbf{x} + \epsilon)^{\mathrm{T}}] = \mathbb{E}[\mathbf{x}\left(\mathbf{W}\mathbf{x}\right)^{\mathrm{T}} + \mathbf{x}\epsilon^{\mathrm{T}}] & (49)\\ &= \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}] + \mathbb{E}[\mathbf{x}]\mathbb{E}[\epsilon] = \mathbb{E}[(\mathbf{x} - \mathbf{0})(\mathbf{x} - \mathbf{0})^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}} + 0 \cdot 0 & (50)\\ &= \mathbf{I}\mathbf{W}^{\mathrm{T}} = \mathbf{W}^{\mathrm{T}}. & (51) \end{aligned}$$

We can then derive the final element of the joint distribution the variance of the observed data,

$$\begin{aligned} \mathbb{E}[(\mathbf{y} - \mathbb{E}[\mathbf{y}])(\mathbf{y} - \mathbb{E}[\mathbf{y}])^{\mathrm{T}}] &= \mathbb{E}[(\mathbf{W}\mathbf{x} + \mu + \epsilon - \mu)(\mathbf{W}\mathbf{x} + \mu + \epsilon - \mu)^{\mathrm{T}}] & (52)\\ &= \mathbb{E}[(\mathbf{W}\mathbf{x} + \epsilon)(\mathbf{W}\mathbf{x} + \epsilon)^{\mathrm{T}}] = \mathbb{E}[\mathbf{W}\mathbf{x}\left(\mathbf{W}\mathbf{x}\right)^{\mathrm{T}} + \mathbf{W}\mathbf{x}\epsilon^{\mathrm{T}} + \epsilon\mathbf{W}\mathbf{x}^{\mathrm{T}} + \epsilon\epsilon^{\mathrm{T}}] & (53)\\ &= \mathbb{E}[\mathbf{W}\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}] + \mathbb{E}[\mathbf{W}\mathbf{x}\epsilon^{\mathrm{T}}] + \mathbb{E}[\epsilon(\mathbf{W}\mathbf{x})^{\mathrm{T}}] + \mathbb{E}[\epsilon\epsilon^{\mathrm{T}}] & (54)\\ &= \mathbf{W}\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}} + \mathbf{W}\mathbb{E}[\mathbf{x}]\mathbb{E}[\epsilon] + \mathbb{E}[\epsilon]\mathbb{E}[\mathbf{x}^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}} + \mathbb{E}[(\epsilon - 0)(\epsilon - 0)^{\mathrm{T}}] & (55)\\ &= \mathbf{W}\mathbf{I}\mathbf{W}^{\mathrm{T}} + \mathbf{W}0 + 0\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I} & (56)\\ &= \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}. & (57) \end{aligned}$$

Now we have all the elements and can write up the final joint distribution as,

$$p(\mathbf{y}, \mathbf{x}|\mathbf{W}) = \mathcal{N}\left(\begin{bmatrix} \mu \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I} & \mathbf{W} \\ \mathbf{W}^{\mathrm{T}} & \mathbf{I} \end{bmatrix}\right), \tag{58}$$

from which we can derive the marginal over the observed data and the conditional distribution over the latent space using the Gaussian identities as,

$$p(\mathbf{y}|\mathbf{W}) = \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}) \tag{59}$$

$$p(\mathbf{x}|\mathbf{y}, \mathbf{W}) = \mathcal{N}(\mathbf{W}^{\mathrm{T}}\left(\mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}\right)^{-1}(\mathbf{y} - \mu), \mathbf{I} - \mathbf{W}^{\mathrm{T}}(\mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I})^{-1}\mathbf{W}). \tag{60}$$

Now we have everything that we need in order to proceed with the inference task. You can approach the above in several different ways but our goal here is that we want to show how this leads to the MDS solution. As it turns out this is the Maximum-Likelihood solution. Or rather it is what is called a Type-II Maximum-Likelihood as we have already marginalised out one of the latent variables namely $\mathbf{X}$.

The maximum-likelihood solution can be reached by taking the derivatives of Eq.59 and taking the stationary point for $\mathbf{W}$ and for σ^2. Its quite a fun exercise to do and provides good training for working with matrix derivatives. If you want to see the full solution have a look in Tipping et al., 1999 in Appendix A. If we do this we will get the following solution,

$$\hat{\mathbf{W}} = \boldsymbol{V}\left(\Lambda - \sigma^2\mathbf{I}\right) \tag{61}$$

$$\hat{\boldsymbol{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d}\lambda_i \tag{62}$$

$$\mathbf{S} = \frac{1}{N-1}\left(\mathbf{Y} - \boldsymbol{\mu}\right)^{\mathrm{T}}\left(\mathbf{Y} - \boldsymbol{\mu}\right) = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}. \tag{63}$$

So the Maximum Likelihood solution to the following model is simply choosing the mapping to be the eigenvectors of the sample covariance matrix $\mathbf{S}$ multiplied by the eigenvalues corrected by the noise variance. If we assume no noise in the data $\sigma^2 \to 0$ then its just the eigenvectors. In many textbooks this is solution is explained as PCA and an interpretation is provided to the algorithm saying something along the lines of. PCA aims to find a *similar* matrix to the sample covariance and choose the linear subspace with maximum variance. This is fine but it misses the point, it does not tell you the assumptions that have been made to reach this solution, it does not show that you have made a clear statement by specifying $p(\mathbf{X})$. When applying PCA to data we should only do so if we believe that the data is normally distributed, otherwise it is the wrong model. Sadly the algorithm does not tell you this. Furthermore, and this is when it becomes really important, if we do not provide the model then it becomes very hard to non-linearise PCA and make assumptions about a general $\mathbf{F}$. Now with this presentation I think all of you can derive the GP-LVM with just a few lines of math, rather than integrating out $\mathbf{X}$ you place a Gaussian process prior over $\mathbf{F}$ and integrate this out. Now you are left with $p(\mathbf{Y} \mid \mathbf{X})$ and if you then optimise $\mathbf{X}$ you have exactly implemented Lawrence, 2005. This is the importance of models and to keep models and algorithms separate.

Equivivalence between MDS and PCA

Now in the linear setting we can derive a simple equvivalence between MDS and PCA. To do so we will start with the algorithmic interpretation of PCA as a method to find the linear subspace through the data that retains the maximal amount of variance. The way that we will show the equivalence is by noting that while MDS diagonalises the $N \times N$ Gram matrix $\mathbf{Y}\mathbf{Y}^{\mathrm{T}}$ PCA diagonalises the $d \times d$ sample matrix $\frac{1}{N-1}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}$[6]. The way we will show the equvivalence is showing the connection between the two diagonalisations and how

[6]Assuming the data is centered

their eigenvectors can be written in terms of each other. We will start from the Gram matrix,

$$\mathbf{G} = \mathbf{Y}\mathbf{Y}^{\mathrm{T}} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}} \tag{64}$$

$$\left(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\right)\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{65}$$

$$\frac{1}{N-1}\mathbf{Y}^{\mathrm{T}}\left(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\right)\mathbf{v}_i = \lambda_i \frac{1}{N-1}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i \tag{66}$$

$$\underbrace{\frac{1}{N-1}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\right)}_{\mathbf{S}}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i = \lambda_i \frac{1}{N-1}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i \tag{67}$$

$$\mathbf{S}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\right) = \frac{\lambda_i}{N-1}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\right) \tag{68}$$

If we now look a bit closer at Eq.68 we can see that this has exactly the behaviour of an eigendecomposition of $\mathbf{S}$ where we would interpret $\frac{\lambda_i}{N-1}$ as the eigenvalue and $\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i$ as the corresponding eigenvector[7]. Importantly as the scaling of the MDS eigenvalue only depends on N the order of the eigenvalues remains the same. There is just one thing missing, the eigenvectors should form an orthonormal basis and if $\mathbf{V}$ does so it is not certain at all that the set $\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i$ does. While we know it will remain orthogonal we need to enforce the constraint that the norm of the eigenvector should be 1.

$$\left(\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\right)^{\mathrm{T}}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\right) = \mathbf{v}_i^{\mathrm{T}}\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i = \lambda_i \tag{69}$$

$$\frac{1}{\sqrt{\lambda_i}}\mathbf{v}_i^{\mathrm{T}}\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\frac{1}{\sqrt{\lambda_i}} = \frac{\lambda_i}{\lambda_i} \tag{70}$$

$$\left(\frac{1}{\sqrt{\lambda_i}}\mathbf{Y}^{\mathrm{T}\mathbf{v}_i}\right)^{\mathrm{T}}\left(\frac{1}{\sqrt{\lambda_i}}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_i\right) = 1 \tag{71}$$

Now we have an orthonormal basis and we know that the eigenvectors of $\mathbf{S}$ can be written in terms of the eigenvectors of $\mathbf{G}$. This shows the equivalence but just to make it extra clear we will look at the embeddings. If we pick the eigenvector corresponding to the largest eigenvalue of the Gram matrix $\mathbf{v}_1$ the first dimension of the embedding that MDS learns is,

$$\mathbf{x}_{MDS} = \mathbf{v}_1\sqrt{\lambda_1}.$$

In PCA the representation is taken by projecting the data onto the eigenvector as,

$$\mathbf{x}_{PCA} = \mathbf{Y}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{v}_1\frac{1}{\sqrt{\lambda_1}}\right) = \underbrace{\left(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\mathbf{v}_1\right)}_{\lambda_i\mathbf{v}_i}\frac{1}{\sqrt{\lambda_1}} = \lambda_1\mathbf{v}_1\frac{1}{\sqrt{\lambda_1}} = \sqrt{\lambda_1}\mathbf{v}_1$$

So the embedding of the two are exactly the same and we have shown the equivalence.[8]

Really this shouldn't be all to surprising, if we come back and think about the rank-nullity theorem and think about the dimensionality of the image of the transformation in the two different scenarios. Lets say that we have data $\mathbf{Y} \in \mathbb{R}^{N\times d}$, so N points in d dimensions. Now if we look at the PCA formulation it works on the sample covariance $\mathbf{S} \in \mathbb{R}^{d\times d}$ so the maximum rank of this matrix is naturally d while for MDS we work on the Gram matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$. However, the maximum rank of this matrix is not n as it is generated from data that only has d degrees of freedom. Therefore its rank is the same as the covariance matrix. This means,

$$\mathrm{Rank}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\right) = \mathrm{Rank}\left(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\right).$$

So why is this equivalence important or at all interesting, well the first thing is that PCA was derived as a model and this is something that we can interpret, we clearly see that the problem is ill-determined and

[7]Its actually the definition of an eigenvalue problem, where we try to solve $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$.

[8]This is admittedly a rather sloppy way to arrive at this conclusion and might not be considered a proof. Another way to do this is to relate the eigenvectors from the two and show that they can be derived from each other.

understand the assumptions that we include in order to reach the solution. We describe, clearly how we believe that data to have been generated and its clear how naive the model really is. When we described MDS it was a bit unclear what the assumption of minimising the frobenious norm actually implied but this is something that we can now take from PCA and translate the assumptions. From this we can say that MDS is also assuming that the data is assumed to be distributed as Gaussian.

Now from an algorithmic point of view their application is slightly different, where MDS is solving an eigenvalue problem that is $N \times N$ PCA solves one that is $D \times D$. This means that if we have the data in vectorial form we would never use MDS and instead we will use the PCA formulation. But as we do not need to have a vectorial input to do MDS it allows us to apply it to scenarios where we do not have access to the representation of the data but only its pairwise similarity.

The important distinction though comes when you try to non-linearise the algorithms. In the MDS setting you do so by coming up with a heuristic to compute the distance along the manifold of the data. In the PCA setting you have a model, a very general model and you can specify a prior over the generative mapping that has non-zero mass for non-linear functions. If you choose a Gaussian process it leads to the GP-LVM. In the latter setting it is very clear what assumptions that you make, the assumptions is over the function space that is relating all the points to each other. In the non-linear MDS setting this is not the case at all, now it is a relationship between individual points that is specified which *implies* a relationship between all the points[9]. This at least to me is much harder to interpret and is likely to be much less robust. In a way I would argue that it is always better to specify a global assumption that leads to local assumptions rather than a local that leads to global[10]. The model allows you to do this while the algorithm is specified directly based on local characteristics.

Summary

In this document we have derived the two most classic dimensionality reduction MDS and PCA and showed that the are indeed leading to the same solution. We took a modelling perspective when we derived PCA and an algorithmic perspective when we did MDS.

One thing that we haven't discussed, but is something that you should be able to extract from what we have derived is why a covariance matrix has to be positive-semi-definite, i.e. that all its eigenvalues have to be non-negative. You can arrive at this conclusion from two directions, one from the Gram matrix and one from the covariance perspective and they both give different insights. Its quite fun to do so its something that might be worthwhile having a look at.

Most likely this document is full of errors so it would be great if you had comments on the things in here and I'll update this. I am notoriously sloppy with notation and math in general so if there is something that you do not get, it is more likely that it is me who has done an error than you misunderstanding.

Acknowledgement

Thanks to Francisco Vargas for pointing out an error in the derivation MDS minimisation.

References

Cox, M and T Cox (Jan. 2008). “Multidimensional scaling”. In: *Handbook of data visualization.*

Hotelling, H (Sept. 1933). “Analysis of a complex of statistical variables into principal components.” In: *Journal of Educational Psychology* 24.6, pp. 417–441.

[9] As the proximity matrix has to be completely filled.

[10] If you want to extend this argument you can take the challenge of understanding what a "Bayesian" Neural Network does compared to a Gaussian processs. Its very hard to interpret the "local" prior that the first implements while for the "global" GP its very natural.

Lawrence, Neil D (2005). “Probabilistic non-linear principal component analysis with Gaussian process latent variable models”. In: *Journal of Machine Learning Research* 6, pp. 1783–1816.

— (2012). “A Unifying Probabilistic Perspective for Spectral Dimensionality Reduction: Insights and New Models”. In: 98888, pp. 1609–1638.

Spearman, Charles (1904). “" General Intelligence," Objectively Determined and Measured”. In: *The American Journal of Psychology* 15.2, pp. 201–292.

Tenenbaum, Joshua B, Vin de Silva, and John C Langford (Dec. 2000). “A Global Geometric Framework for Nonlinear Dimensionality Reduction”. In: *Science* 290.5500, pp. 2319–2323.

Tipping, Michael E and Christopher M Bishop (1999). “Probabilistic principal component analysis”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.3, pp. 611–622.

Weinberger, Kilian Q, Fei Sha, and Lawrence K Saul (July 2004). “Learning a kernel matrix for nonlinear dimensionality reduction”. In: *International Conference on Machine Learning*. New York, New York, USA: ACM, pp. 106–113.