

SOME STUFF FOR HOME

GASPARD BUMA

1. NOTATIONS

In the following discussion the following typographic notations has been used:

- *Lower case bold face* means vector (ex. $\boldsymbol{\mu}$, \boldsymbol{x}). All vectors are row vectors like $[x_1 \ x_2]$ not $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.
- *Upper case bold face* means matrix (ex. \boldsymbol{A} , \boldsymbol{S} , $\boldsymbol{\Sigma}$).
- The transpose of a matrix is noted \boldsymbol{A}^T , not \boldsymbol{A}' .
- \boldsymbol{x}' , \boldsymbol{A}' denotes a vector/matrix that has been transformed.

2. VARIANCE OF A VALUE

The **variance** (expressed with the symbol σ^2 or σ_S^2) measures the dispersion of a set of values \boldsymbol{S} from the mean value μ or expected value $E(\boldsymbol{S})$. The mean value is :

$$\mu = E(\boldsymbol{S}) = \frac{1}{l} \sum_1^l s_l$$

l being the number of elements in the set. The variance can be computed as :

$$\sigma^2 = E[(\boldsymbol{S} - \mu)(\boldsymbol{S} - \mu)] = \frac{1}{l} \sum_1^l (s_l - \mu)^2$$

For example, given the set of values $\boldsymbol{S} = 1, 4, 3, 6$, the mean value μ is :

$$\mu = \frac{1 + 4 + 3 + 6}{4} = 3.5,$$

and the variance σ^2 :

$$\begin{aligned} \sigma^2 &= (1 - 3.5)^2 + (4 - 3.5)^2 + \\ &\quad \frac{(3 - 3.5)^2 + (6 - 3.5)^2}{4} = \frac{13}{4} = 3.25 \end{aligned}$$

3. VARIANCE-COVARIANCE MATRIX

The **variance-covariance matrix** is the generalisation to greater dimensions of the variance seen before. Imagine we have a set \boldsymbol{S} of vectors of dimension n and we want to compute the variance of these vectors. We must first compute the expected value $E(\boldsymbol{S})$ which is the mean vector $\boldsymbol{\mu}$:

$$\begin{aligned} E(\boldsymbol{S}) = \boldsymbol{\mu} &= [\mu_1 \dots \mu_n] \\ &= \frac{1}{l} \begin{bmatrix} [s_{11} \dots s_{1n}] \\ + \\ \vdots \\ + \\ [s_{l1} \dots s_{ln}] \end{bmatrix} \end{aligned}$$

We can build a matrix $\boldsymbol{\Theta}$ (theta) of size (l, n) from the vector $\boldsymbol{\mu}$ (mu) :

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\mu} \\ \vdots \\ \boldsymbol{\mu} \end{bmatrix}$$

From the definition of the variance, we have

$$\boldsymbol{\Sigma} = E[(\boldsymbol{S} - \boldsymbol{\Theta})(\boldsymbol{S} - \boldsymbol{\Theta})^T]$$

thus each entry in the variance-covariance matrix $\boldsymbol{\Sigma}$ (sigma) is :

$$\Sigma_{ij} = \frac{\sum_{k=1}^l (s_{ki} - \mu_i)(s_{kj} - \mu_j)}{l}.$$

For example, given the set \boldsymbol{S} formed with three experiments measuring two values :

$$\boldsymbol{S} = \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 1 \end{bmatrix},$$

we can compute the expected value :

$$\boldsymbol{\mu} = \frac{1}{3} \left[\begin{array}{c} \left(\begin{array}{c} 1 \\ + \\ 0 \\ + \\ 2 \end{array} \right) \left(\begin{array}{c} 1 \\ + \\ 4 \\ + \\ 1 \end{array} \right) \end{array} \right] = [1 \quad 2].$$

To find the variance-covariance matrix, let us first compute the *mean deviation form* of \mathbf{S} where we remove the mean value from each experiment in the set :

$$\mathbf{T} = \mathbf{S} - \boldsymbol{\Theta} = \begin{bmatrix} (1-1) & (1-2) \\ (0-1) & (4-2) \\ (2-1) & (1-2) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 2 \\ 1 & -1 \end{bmatrix}.$$

We can now find $\boldsymbol{\Sigma}$:

$$\begin{aligned} \boldsymbol{\Sigma} &= E[\mathbf{T}^T \mathbf{T}] = \frac{1}{n-1} [\mathbf{T}^T \mathbf{T}] \\ &= E \left[\begin{bmatrix} 0 & -1 & 1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 2 \\ 1 & -1 \end{bmatrix} \right] \\ &= E \left[\begin{array}{cc} 0*0 + (-1)*(-1) + 1*1 & \dots \\ -1*0 + 2*(-1) + (-1)*1 & \dots \end{array} \right] \\ &= E \left[\begin{array}{cc} 2 & -3 \\ -3 & 6 \end{array} \right] = \frac{1}{2} \begin{bmatrix} 2 & -3 \\ -3 & 6 \end{bmatrix} = \begin{bmatrix} 1 & -\frac{3}{2} \\ -\frac{3}{2} & 3 \end{bmatrix} \end{aligned}$$

From the result above, 1 is the variance along x , 3 is the variance along y and $-\frac{3}{2}$ is the covariance between the two dimensions. There are proofs on why we should use $\frac{1}{n-1}$ for the correct normalization factor instead of the straight-forward $\frac{1}{n}$ but this is beyond the scope of my competences...

4. MAHALANOBIS DISTANCE

Let us compute the **Mahalanobis distance** from a vector \mathbf{t} to the set \mathbf{S} with the variance-covariance matrix $\boldsymbol{\Sigma}$ and the mean value $\boldsymbol{\mu}$:

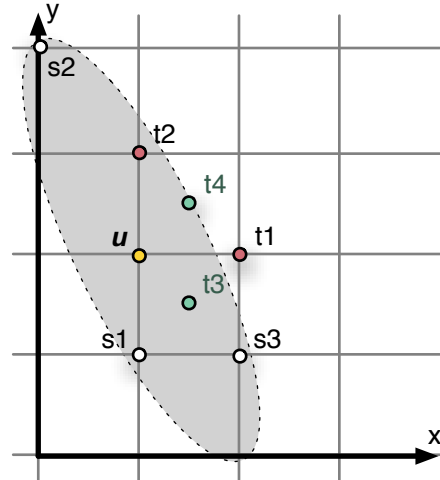
$$D_M(\mathbf{t}) = \sqrt{(\mathbf{t} - \boldsymbol{\mu}) \boldsymbol{\Sigma}^{-1} (\mathbf{t} - \boldsymbol{\mu})^T}$$

If we remove the covariances from the matrix $\boldsymbol{\Sigma}$, we have a diagonal matrix whose entries are the variances (standard deviations) for each dimension (1 and 3 in the

example above). The Mahalanobis distance becomes the **normalized Euclidean distance** :

$$D_{nE}(\mathbf{t}) = \sqrt{\sum_{i=1}^n \frac{(t_i - \mu_i)^2}{\sigma_i^2}}.$$

From the example we used above we compute the different distances for two vectors \mathbf{t}_1 and \mathbf{t}_2 :



$$\mathbf{t}_1 = [2 \quad 2]$$

$$\mathbf{t}_2 = [1 \quad 3]$$

$$\boldsymbol{\mu} = [1 \quad 2]$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & -\frac{3}{2} \\ -\frac{3}{2} & 3 \end{bmatrix}$$

The **Euclidean distance** d is the same for both points, even though it is visible that \mathbf{t}_1 is more “out of the way” than \mathbf{t}_2 :

$$\begin{aligned} d_1 &= \sqrt{\sum_{i=1}^n (t_i - \mu_i)^2} \\ &= \sqrt{(2-1)^2 + (2-2)^2} = 1 \\ d_2 &= \sqrt{(1-1)^2 + (3-2)^2} = 1 \end{aligned}$$

The **normalized Euclidean distance** d_{nE} should show that \mathbf{t}_2 matches the set \mathbf{S}

better than \mathbf{t}_1 :

$$d_{1nE} = \sqrt{\frac{(2-1)^2}{1} + \frac{(2-2)^2}{3}} = 1$$

$$d_{2nE} = \sqrt{\frac{(1-1)^2}{1} + \frac{(3-2)^2}{3}} = \frac{1}{\sqrt{3}} \approx 0.6.$$

This is better. We can see that \mathbf{t}_1 is penalized for being out of the way in the wrong direction. The “penalty” for being badly adjusted along the x axis is 1 when it is only $\frac{1}{3}$ along the y axis.

Let’s now see what the Mahalanobis distance tells us. First we need to compute Σ^{-1} :

$$\Sigma^{-1} = \frac{1}{(1 * 3 - -\frac{3}{2} * -\frac{3}{2})} \begin{bmatrix} 3 & \frac{3}{2} \\ \frac{3}{2} & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & \frac{4}{3} \end{bmatrix}$$

We can now find the distances :

$$d_{1M} = \sqrt{(\mathbf{t}_1 - \boldsymbol{\mu})\Sigma^{-1}(\mathbf{t}_1 - \boldsymbol{\mu})^T}$$

$$= \sqrt{\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 2 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}}$$

$$= \sqrt{\begin{bmatrix} 4 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}} = 2$$

$$d_{2M} = \sqrt{\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 2 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}}$$

$$= \sqrt{\begin{bmatrix} 2 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}} = \frac{2}{\sqrt{3}} \approx 1.2$$

These distances classify the two vectors in the exactly the same way as the normalized Euclidean distances. Why is that so ? The reason is that our vectors are always away from the mean value in a single direction at a time, thus the “covariance” part of the error is never used.

What happens with the two points \mathbf{t}_3 and \mathbf{t}_4 ? From the first look at the picture we can see that their distance to the

mean value is the same :

$$d_3 = \frac{1}{\sqrt{2}} \approx 0.7$$

$$d_4 = \frac{1}{\sqrt{2}} \approx 0.7$$

Their normalized Euclidean values are also the same as they differ from the mean value exactly in the same way along x and y :

$$d_{3nE} = \frac{1}{\sqrt{3}} \approx 0.6$$

$$d_{4nE} = \frac{1}{\sqrt{3}} \approx 0.6$$

Now comes the Mahalanobis distances :

$$d_{3M} = \frac{1}{\sqrt{3}} \approx 0.6$$

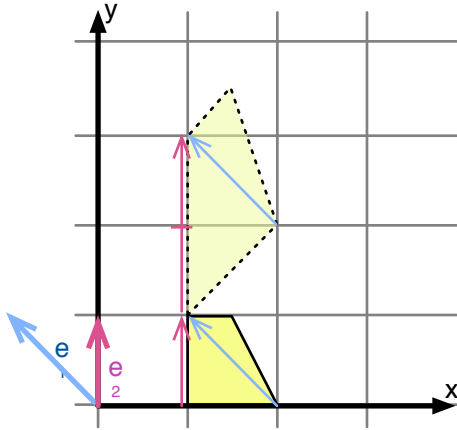
$$d_{4M} = \frac{\sqrt{14}}{\sqrt{6}} \approx 1.5$$

This type of distance calculation is the only one that reflect the difference between the two vectors \mathbf{t}_3 and \mathbf{t}_4 and classifies \mathbf{t}_4 as being further “away” then \mathbf{t}_3 , like our eyes tell us from a quick look at the picture !

5. EIGENVALUES

Before we study *Principal Component Analysis*, we need some understanding of eigenvalues. Imagine we have a transformation matrix \mathbf{P} :

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}.$$



What are the vectors in this transformation that keep their direction (are not rotated)? Mathematically speaking we want to find \mathbf{e} such that :

$$\mathbf{e}' = \mathbf{e} * \mathbf{P} = \lambda \mathbf{e}$$

We can see from the picture that \mathbf{y} satisfy this as \mathbf{y}' is just $2 * \mathbf{y}$. Such vectors are called *eigenvectors* and their corresponding lambda values *eigenvalues*. To find these vectors, we can rewrite the equation above :

$$\mathbf{e}' = \mathbf{e} * \mathbf{P} = \lambda \mathbf{e}$$

as

$$\mathbf{e} * \mathbf{P} - \lambda \mathbf{e} = 0$$

or

$$\mathbf{e}(\mathbf{P} - \lambda \mathbf{I}) = 0$$

We know that an equation of the form :

$$\mathbf{x}\mathbf{A} = 0$$

has only the trivial solution $\mathbf{x} = \mathbf{0}$ if the determinant of \mathbf{A} is not null. We can therefore find the non-trivial solutions to the equation above by computing the determinant when the latter is null. This equation is called the *characteristic equation* of \mathbf{P} :

$$\det(\mathbf{P} - \lambda \mathbf{I}) = 0$$

We can now calculate the values of λ :

$$\det\left(\begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

$$\det\left(\begin{bmatrix} 1 - \lambda & 1 \\ 0 & 2 - \lambda \end{bmatrix}\right) = 0$$

$$(1 - \lambda)(2 - \lambda) - 1 * 0 = 0$$

$$(1 - \lambda)(2 - \lambda) = 0$$

which has the solutions

$$\lambda_1 = 1, \lambda_2 = 2$$

The corresponding *eigenvectors* can be found by substituting λ_1 and λ_2 in $\mathbf{e}(\mathbf{P} - \lambda \mathbf{I}) = \mathbf{0}$:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 - \lambda & 1 \\ 0 & 2 - \lambda \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{bmatrix} 1 - 1 & 1 \\ 0 & 2 - 1 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$$

$$x_1 + y_1 = 0$$

$$\mathbf{e}_1 = \begin{bmatrix} t & -t \end{bmatrix}$$

and for λ_2 :

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} \begin{bmatrix} 1 - 2 & 1 \\ 0 & 2 - 2 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix} = 0$$

$$-x_2 = 0$$

$$x_2 = 0$$

$$\mathbf{e}_2 = \begin{bmatrix} 0 & t \end{bmatrix} = \mathbf{y}$$

If we look at our transformation picture again, we see that the relation in the direction \mathbf{e}_1 is kept and the relation in the direction of \mathbf{e}_2 is just doubled. This reflects the eigenvalues 1 and 2 with the directions of the eigenvectors \mathbf{e}_1 and \mathbf{e}_2 .

Before moving on to the following section, we need to establish some properties of *symmetric* matrices. The first one is that such a matrix \mathbf{A} can be expressed by the multiplication of a diagonal matrix \mathbf{D} (all

entries not on the diagonal are $\mathbf{0}$) and an invertible matrix \mathbf{E} :

$$\mathbf{A} = \mathbf{E}^T \mathbf{D} \mathbf{E}.$$

Because \mathbf{A} is symmetric, the transpose \mathbf{A}^T of \mathbf{A} (mirror across the diagonal) is the same as \mathbf{A} , we can therefore write :

$$\begin{aligned} \mathbf{A}^T &= (\mathbf{E}^T \mathbf{D} \mathbf{E})^T \\ &= \mathbf{E}^T (\mathbf{E}^T \mathbf{D})^T \\ &= \mathbf{E}^T \mathbf{D}^T \mathbf{E}^{TT} \end{aligned}$$

now we simply replace \mathbf{D}^T by \mathbf{D} since a diagonal matrix is equivalent to its transpose and we replace \mathbf{E}^{TT} by \mathbf{E} since transposing twice is equivalent to doing nothing. We get

$$\mathbf{A}^T = \mathbf{E}^T \mathbf{D} \mathbf{E} = \mathbf{A}.$$

This proves that any matrix that can be expressed as a product of an invertible matrix \mathbf{E} and a diagonal matrix \mathbf{D} is indeed symmetric. We should now prove the reverse : that any symmetric matrix can be expressed as stated above. This proof is called the *spectral theorem*. While trying to understand it, I fell across strange names like *Hermitian matrices*, *Schur decomposition* and *Hilbert spaces*. It also involved matrix calculations with complex numbers. That was too much me. Sorry. For those interested, I found a proof which you can consult on the web¹.

Anyway, let's pass on to the interesting part : the matrix \mathbf{E} used in the little talk above is in fact a matrix of the orthonormal eigenvectors of \mathbf{A} (those vectors that do not rotate when transformed by \mathbf{A}). The proof is interesting but quite long, you can safely pass on to the section on *Principal Component Analysis* if you do not have fun with letters in capital bold.

Please note that the following proofs are a rewrite of the first appendix of the tutorial on PCA by Jonathon Shlens².

First, lets build this matrix \mathbf{E} from the eigenvectors of \mathbf{A} :

$$\mathbf{E} = \begin{bmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_n \end{bmatrix}.$$

Now let \mathbf{D} be a diagonal matrix with the corresponding eigenvalues of \mathbf{A} :

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 & \dots \\ 0 & \lambda_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

We first try to prove that $\mathbf{E} \mathbf{A} = \mathbf{D} \mathbf{E}$:

$$\begin{aligned} \mathbf{E} \mathbf{A} &= \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \end{bmatrix} \mathbf{A} = \begin{bmatrix} \mathbf{e}_1 \mathbf{A} \\ \mathbf{e}_2 \mathbf{A} \\ \vdots \end{bmatrix} \\ \mathbf{D} \mathbf{E} &= \begin{bmatrix} \lambda_1 & 0 & \dots \\ 0 & \lambda_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \mathbf{E} = \begin{bmatrix} \lambda_1 \mathbf{e}_1 \\ \lambda_2 \mathbf{e}_2 \\ \vdots \end{bmatrix}. \end{aligned}$$

For $\mathbf{E} \mathbf{A}$ to be equal to $\mathbf{D} \mathbf{E}$, we must show that $\mathbf{e}_i \mathbf{A} = \lambda_i \mathbf{e}_i$ for all i . Remember that an eigenvector is a vector that is scaled onto itself by the transformation, that is $\mathbf{e}' = \lambda \mathbf{e}$. This is exactly what the result of our equation states, thus the entries of \mathbf{E} are the eigenvectors of \mathbf{A} if $\mathbf{E} \mathbf{A} = \mathbf{D} \mathbf{E}$.

To finish the proof, we just need to show that $\mathbf{E} \mathbf{A} = \mathbf{D} \mathbf{E}$ implies $\mathbf{A} = \mathbf{E}^T \mathbf{D} \mathbf{E}$. By multiplying each side by \mathbf{E}^{-1} , we obtain $\mathbf{A} = \mathbf{E}^{-1} \mathbf{D} \mathbf{E}$. If we could show that $\mathbf{E}^{-1} = \mathbf{E}^T$, we would have proven (5). We know

¹(<http://users.utu.fi/jkari/compression/kltaddition.pdf>) ²(www.sn1.salk.edu/~shlens/pub/notes/pca.pdf)

that $\mathbf{E}^{-1} = \mathbf{E}^T$ if \mathbf{E} is an orthogonal matrix since :

$$\begin{aligned} \mathbf{E}\mathbf{E}^T &= \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \end{bmatrix} [\mathbf{e}_1^T \quad \mathbf{e}_2^T \quad \dots] \\ &= \begin{bmatrix} \mathbf{e}_1\mathbf{e}_1^T & \mathbf{e}_1\mathbf{e}_2^T & \dots \\ \mathbf{e}_2\mathbf{e}_1^T & \mathbf{e}_2\mathbf{e}_2^T & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \dots \\ 0 & 1 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \\ &= \mathbf{I} \end{aligned}$$

because each vectors in an orthogonal matrix is \perp to other vectors, thus the dot product is zero except for the diagonal entries where it is equal to the squared norm of the vector. If we choose our eigenvectors so that their norm is 1, we obtain the identity matrix \mathbf{I} . We just need to show that \mathbf{E} is an orthogonal matrix and pulling the thread out of this maze, we prove that a symmetric matrix can be expressed by (5). We want to know if $(\vec{e}_i)\vec{e}_j = 0$ for all $i \neq j$. Let λ_i and λ_j be two distinct eigenvalues.

$$\begin{aligned} \lambda_i \mathbf{e}_i \cdot \mathbf{e}_j &= \lambda_i \mathbf{e}_i \mathbf{e}_j^T \\ &= \mathbf{e}_i \mathbf{A} \mathbf{e}_j^T \\ &= \mathbf{e}_i (\mathbf{e}_j \mathbf{A}^T)^T \\ &= \mathbf{e}_i (\mathbf{e}_j \mathbf{A})^T \\ &= \mathbf{e}_i (\lambda_j \mathbf{e}_j^T) \\ \lambda_i \mathbf{e}_i \cdot \mathbf{e}_j &= \mathbf{e}_i \cdot \lambda_j \mathbf{e}_j \end{aligned}$$

Thus $(\lambda_i - \lambda_j) \mathbf{e}_i \cdot \mathbf{e}_j = 0$. Since we have conjectured that $\lambda_i \neq \lambda_j$, it must be that \mathbf{e}_i and \mathbf{e}_j are orthonormal (their dot product is 0). Therefore $\mathbf{E}^{-1} = \mathbf{E}^T$ and we can write :

$$\mathbf{A} = \mathbf{E}^T \mathbf{D} \mathbf{E}.$$

All this work, just to reexpress an arbitrary symmetric matrix \mathbf{A} . . . But this result is the ground on which the next section is built.

6. PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis is a way to extract the “most prominent features” of a data set. It assumes that the data has a good signal to noise ratio which means that *great variance* in the data is a synonym for *relevant feature* and not *big noise*.

In all the calculations below, we need to use the *variations* of the measures. We will thus remove the mean values from our data set and use this new set called the *mean deviation form* of \mathbf{S} :

$$\mathbf{T} = \mathbf{S} - \begin{bmatrix} \mu \\ \vdots \\ \mu \end{bmatrix}.$$

In order to extract the “most prominent features” of our data, a good idea would be to look at the directions in which the data varies most. Is it along the \mathbf{x} axis ? The \mathbf{y} axis ? Somewhere in between ?

Recall from the previous sections that the variance-covariance matrix Σ displays variances along the diagonal and covariances off-diagonal. We can use this matrix to extract directions in which the data varies most :

$$\Sigma = \frac{1}{n-1} \mathbf{T}^T \mathbf{T}.$$

By looking at the diagonals of Σ , we pick the greatest absolute value and there we are, we found our direction ($\mathbf{3} \Rightarrow \mathbf{y}$). Is that it ? No, this solution misses the fact that the main direction might not follow one of the axis (dimension) of our recorded data (the ellipsoid is not vertical nor horizontal). What we need is to apply a transformation (rotation + stretch) to our data \mathbf{T} into an new basis \mathbf{P} prior to looking at the variance-covariance matrix, so that the

principal components align with the new basis' axis.

$$\mathbf{T}' = \mathbf{T}\mathbf{P}$$

$$\mathbf{\Sigma}' = \frac{1}{n-1} \mathbf{T}'^T \mathbf{T}'$$

Our goal is thus to find a new basis \mathbf{P} that gives us great absolute values along the diagonals of $\mathbf{\Sigma}'$ and small absolute values off-diagonal (small redundancy). If possible, it would be great to see large variances along the top of the diagonal which would mean we could just cut off the right-most part of the matrix \mathbf{T}' and still have most of the information on the varying signal. To summarize :

- Find a new basis \mathbf{P} that reduces the redundancy of the signals (low covariance).
- The $\mathbf{\Sigma}'$ matrix for \mathbf{T}' should ideally be a diagonal matrix with very high values at the top and very small at the bottom, thus showing signals in the first dimensions and pure noise in the last ones.
- We could then choose to remove the columns of \mathbf{T}' that have greater indices than the number of dimensions we want to keep thus reducing the dimensionality of our data.

A simple way to find the solution would be :

- (1) Find the direction \mathbf{e}_1 that shows the greatest variance in the data \mathbf{T} . Normalize this vector. This is our first basis vector.
- (2) Find another normalized vector showing the greatest variance that is orthogonal to those already found ($\forall \mathbf{b} \in \{\text{found vectors}\}, \mathbf{e}_n \perp \mathbf{b}$).
- (3) Repeat until we have n basis vectors (n is the size of vector space).

The resulting ordered set of vectors are the *principal components* of our data set \mathbf{T} .

7. COMPUTING PCA

Mathematically stated, what we are looking for is a diagonalized version of $\mathbf{\Sigma}'$:

$$\begin{aligned} \mathbf{\Sigma}' &= \frac{1}{n-1} \mathbf{T}'^T \mathbf{T}' \\ &= \frac{1}{n-1} (\mathbf{T}\mathbf{P})^T (\mathbf{T}\mathbf{P}) \\ &= \frac{1}{n-1} \mathbf{P}^T \mathbf{T}^T \mathbf{T} \mathbf{P} \end{aligned}$$

Does this look familiar to you ? Recall all our lengthy discussion on symmetrical matrices ? Well, $\mathbf{T}^T \mathbf{T}$ is such a matrix. Lets replace it by \mathbf{A} . We get :

$$\mathbf{\Sigma}' = \frac{1}{n-1} \mathbf{P}^T \mathbf{A} \mathbf{P}$$

Now what happens if we set $\mathbf{P} = \mathbf{E}^T$ and replace \mathbf{A} by $\mathbf{E}^T \mathbf{D} \mathbf{E}$ from equation (5) ?

$$\begin{aligned} \mathbf{\Sigma}' &= \frac{1}{n-1} \mathbf{E} \mathbf{A} \mathbf{E}^T \\ &= \frac{1}{n-1} \mathbf{E} (\mathbf{E}^T \mathbf{D} \mathbf{E}) \mathbf{E}^T \\ &= \frac{1}{n-1} \mathbf{I} \mathbf{D} \mathbf{I} \\ &= \frac{1}{n-1} \mathbf{D} \end{aligned}$$

We have just found the transformation that diagonalizes the variance-covariance matrix. It's the transpose of the matrix of eigenvectors of $\mathbf{T}^T \mathbf{T}$!

$$\mathbf{P} = \mathbf{E}^T = [\mathbf{e}_1^T \quad \mathbf{e}_2^T \quad \dots]$$

where \mathbf{e}_i is i th eigenvector of $\mathbf{T}^T \mathbf{T}$.

8. APPLYING PCA TO \mathbf{S}

Using the same sample data as above, we imagine that the set of points \mathbf{S} represents the measures of a moving element. The object actually moves only along the ellipsoid axis $\mathbf{e} = [-1 \quad 2]$, the deviations from this axis is just noise. The goal of PCA is to re-express our set of data in a new basis that reflects this behaviour (data along \mathbf{e} , noise

along the other direction). The naive basis (in which we recorded the data) is :

$$\mathbf{B} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

We will use PCA to find a new basis \mathbf{P} that is a linear combination of the original basis \mathbf{B} and in which we will only look at the data along the greatest axis of the ellipsoid (where the data exhibits the greatest variance). In this new basis, the set of data \mathbf{S} becomes :

$$\mathbf{S}' = \mathbf{S}\mathbf{P}$$

The new basis should optimise signal to noise ratio if we consider the first dimension as representing the signal and the second the noise :

$$SNR = \frac{\sigma_{signal}^2}{\sigma_{noise}^2}$$

Before delving any further, we must first transform our data into its *mean deviation form* \mathbf{T} .

$$\mathbf{T} = \mathbf{S} - \begin{bmatrix} \boldsymbol{\mu} \\ \vdots \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 2 \\ 1 & -1 \end{bmatrix}.$$

We can now easily compute the actual SNR :

$$SNR = \frac{\sum_{k=1}^l T_{k1}^2}{\sum_{k=1}^l T_{k2}^2} = \frac{0 + 1 + 1}{1 + 4 + 1} = \frac{1}{3}$$

How do we find the new basis that gives us the greatest SNR ? Expressed in another way, we could say : how do we find the basis that shows greatest variance in the first dimension and mostly noise in the second ? From the preceding section, we know how to build \mathbf{P} . Let's do it with our data. First

we need to find the symmetric matrix \mathbf{A} :

$$\begin{aligned} \mathbf{A} = \mathbf{T}^T \mathbf{T} &= \begin{bmatrix} 0 & -1 & 1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 2 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 2 & -3 \\ -3 & 6 \end{bmatrix} \end{aligned}$$

Now we must find the eigenvalues of this matrix :

$$\begin{aligned} \det(\mathbf{A} - \lambda \mathbf{I}) &= 0 \\ \det \left(\begin{bmatrix} 2 - \lambda & -3 \\ -3 & 6 - \lambda \end{bmatrix} \right) &= 0 \\ (2 - \lambda)(6 - \lambda) - 9 &= 0 \\ \lambda^2 - 8\lambda + 3 &= 0 \\ \lambda_1 \approx 7.6, \lambda_2 \approx 0.4. \end{aligned}$$

The corresponding eigenvectors are :

$$\begin{aligned} [x \quad y] \begin{bmatrix} 2 - \lambda & -3 \\ -3 & 6 - \lambda \end{bmatrix} &= 0 \\ \begin{cases} x = \frac{3}{2 - \lambda} t \\ y = t \end{cases} \end{aligned}$$

We want normed vectors so we add the rule $x^2 + y^2 = 1$ and we get :

$$\begin{aligned} \frac{9t^2}{(2 - \lambda)^2} + t^2 &= 1 \\ t &= \frac{2 - \lambda}{\sqrt{9 + (2 - \lambda)^2}} \\ t_1 &\approx -0.9 \\ t_2 &\approx 0.5 \\ \mathbf{E} &\approx \begin{bmatrix} 0.5 & -0.9 \\ 0.9 & 0.5 \end{bmatrix}. \end{aligned}$$

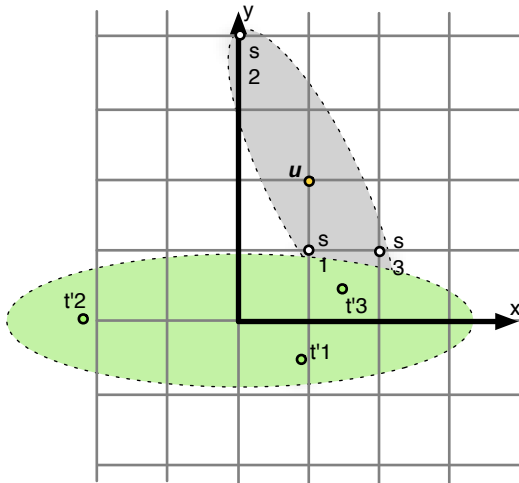
Now that we have our transformation matrix \mathbf{P}

$$\mathbf{P} = \mathbf{E}^T = \begin{bmatrix} 0.5 & 0.9 \\ -0.9 & 0.5 \end{bmatrix}.$$

we can transform our data \mathbf{T} into \mathbf{T}' :

$$\begin{aligned} \mathbf{T}' = \mathbf{T}\mathbf{P} &\approx \begin{bmatrix} 0 & -1 \\ -1 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0.5 & 0.9 \\ -0.9 & 0.5 \end{bmatrix} \\ &\approx \begin{bmatrix} 0.9 & -0.5 \\ -2.2 & 0.1 \\ 1.4 & 0.4 \end{bmatrix} \end{aligned}$$

With this new transformation, we should have great variance along the first dimension and a very small one in the second (that was our goal). By looking at the values, this seems to be the case. Let's see with a picture.



Just to make sure our work was correct, let's compute the variance-covariance matrix Σ' :

$$\Sigma' = \frac{1}{n-1} \mathbf{T}'^T \mathbf{T}' \approx \begin{bmatrix} 7.6 & 0 \\ 0 & 0.4 \end{bmatrix}.$$

It works ! And by the way, this is \mathbf{D} which is the diagonal matrix made out of the eigenvalues of $\mathbf{A} = \mathbf{T}^T \mathbf{T}$ (7.6 and 0.4).

To finish our goal of dimensionality reduction, we can build another version of the matrix \mathbf{P} that automatically strips off the “noisy” dimensions, only keeping the principal components of our data :

$$\mathbf{Q} = \mathbf{P} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -0.9 \end{bmatrix}.$$

And before closing this section, we need to compute our new signal to noise ratio :

$$SNR = \frac{\sum_{k=1}^l T'_{k1}{}^2}{\sum_{k=1}^l T'_{k2}{}^2} \approx \frac{7.6}{0.4} = 19.3.$$

By just looking at the \mathbf{y} axis, we would have obtained 3. Applying the transformation \mathbf{P} made the signal much easier to analyse.

That's it (I'm glad I made it this far) !

8.1. summary on PCA. PCA can be used as a filter to reduce the dimensionality of the data and ease the computation of the **Mahalanobis distance** as this distance becomes the **normalized Euclidean distance** with the new transformation since the covariances have been removed (Σ' is a diagonal matrix). The calculation of the transformation matrix \mathbf{P} involves the following computations :

- Find the mean vector $\boldsymbol{\mu}$.
- Subtract this vector to each vector of the training set \mathbf{S} to obtain the *mean deviation form* \mathbf{T} .
- Find the eigenvectors of the symmetric matrix $\mathbf{T}^T \mathbf{T}$, sorted by decreasing eigenvalues.
- Build the transformation matrix $\mathbf{P} = [\mathbf{e}_1^T \ \mathbf{e}_2^T \ \dots \ \mathbf{e}_k^T]$ with $k \leq n$ to keep the k most relevant dimensions of the transformed data.
- Apply this transformation \mathbf{P} to any test vector \mathbf{t} before computing the normalized Euclidean distance to the mean value $\boldsymbol{\mu}$.
- The smaller the distance, the bigger the probability that \mathbf{t} belongs to the set \mathbf{S} .

Among all the steps above, two are problematic for large matrices: finding the eigenvalues and eigenvectors. These steps imply solving the *characteristic polynomial* of \mathbf{A} .

If our data contains 12 measures x 100 samples per row, the dimension of our polynomial is 1200 ! To find the eigenvector we have to solve n times a system of n-1 equations. This can become daunting. That was for the bad news. The good news is that these calculations must only be done on the training set, not live. The even better news is that there exists a library written in Fortran called *LAPACK* that can do the job as fast as the most clever people could make it. Thanks.

8.2. **notes on PCA.** For a deeper understanding of PCA and it's limits, one may have a look at :

- *kernel PCA* to apply a non-linear filter prior to applying PCA for problems where linearity is an issue.
- *ICA* for sets of data whose distribution probabilities are not exponential (Gaussian, Exponential, etc).
- *Central Limit Theorem* on why PCA should work most of the time in real world as probabilities usually **are** gaussian.

9. MOVEMENT RECOGNITION

From the knowledge we have gathered above, we can now try to find a way to compare the movements. Imagine we have 3 movements (classes) labeled a , b and c . Each movement has been recorded several times. We create the training sets \mathbf{S}_a , \mathbf{S}_b and \mathbf{S}_c with the recordings of the different movements. Each move is made of 2 measures recorded for 10 samples. We write it as a row vector of dimension 20 (\mathbf{s}_{ai}).

10. FILTERING (PCA)

We start by computing the mean vector for each movement. We will note the number of recordings for the current movement

r :

$$\boldsymbol{\mu}_a = \frac{1}{r} \sum_{k=1}^r [\mathbf{s}_{a1k} \quad \mathbf{s}_{a2k} \quad \dots \quad \mathbf{s}_{a20k}]$$

We do this for each class a , b and c . We then build a matrix out of these prototypes :

$$\mathbf{S} = \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \\ \boldsymbol{\mu}_c \end{bmatrix}$$

We now find the mean deviation form of \mathbf{S} after finding the mean value of all the prototypes ($\boldsymbol{\mu}$) :

$$\mathbf{T} = \begin{bmatrix} \boldsymbol{\mu}_a - \boldsymbol{\mu} \\ \boldsymbol{\mu}_b - \boldsymbol{\mu} \\ \boldsymbol{\mu}_c - \boldsymbol{\mu} \end{bmatrix}$$

Now that we have found the *least varying parts* of each movement (the “platonic idea”) by computing the average, we will find a transformation that shows the *most varying parts* **between** these movements. We apply PCA on the matrix \mathbf{T} and find the transformation matrix \mathbf{P} .

$$\mathbf{P} = \mathbf{E}^T \mathbf{I}_{reduc}$$

Where \mathbf{I}_{reduc} is a truncated identity matrix to reduce the dimensionality of the data to the principal components.

11. RECOGNITION

To label new vectors, we could use the Mahalanobis distance or Euclidean distance in the transformed vector space produced by PCA. This is the first solution discussed below. The other approach is to use *Support Vector Machines*. This is the solution which we will try to use if our hardware is fast enough since it produces better results.

11.1. **Distances.** We now have two options. Either we use the information contained in the eigenvalues (matrix \mathbf{D}) to compute the normalized Euclidean distances between our test vector and the prototypes in the new coordinate system defined by the transformation \mathbf{P} or we map all our training data

into the new coordinate system and compute the Mahalanobis distance to each set. This last solution loses some information from the *differences between the different classes* a , b and c but takes into account what is noise and what is a signal in each movement. Since we have already included the information on what makes the differences between the classes when we chose to remove the dimensions with least variance, the second solution seems to keep most of the information contained in the training data.

Transforming each training set into the new coordinate system is done by

$$\begin{aligned} S'_a &= S_a P \\ S'_b &= S_b P \\ S'_c &= S_c P \end{aligned}$$

We now compute the variance-covariance for each of these matrices :

$$\begin{aligned} \Sigma'_a &= \frac{1}{r-1} S'^T_a S'_a \\ \Sigma'_b &= \frac{1}{r-1} S'^T_b S'_b \\ \Sigma'_c &= \frac{1}{r-1} S'^T_c S'_c \end{aligned}$$

Normalization of the values in these matrices ?

$$\Sigma'_{aij} = \frac{\Sigma'_{aij}}{\sum_{k=1}^3 \Sigma'_{kij}}$$

To label an arbitrary movement \mathbf{t} , we find the minimal Mahalanobis distance to the sets in the new coordinate system.

$$\begin{aligned} \mathbf{t}'_k &= \mathbf{t}' - \boldsymbol{\mu}'_k \\ &= \mathbf{t}P - \boldsymbol{\mu}'_k \end{aligned}$$

$$\arg \min_k D_M(\mathbf{t}) = \arg \min_k \sqrt{\mathbf{t}'_k \Sigma'^{-1}_k \mathbf{t}'_k{}^T}$$

The value returned by $\arg \min (k)$ is the label for the test movement.

11.2. Support Vector Machines. *Support Vector Machines* is a new mathematical tool for pattern recognition. Basically, it involves mapping the data into a new vector space (once more) with higher dimensions this time. The goal is to find a vector space in which the data is linearly separable (you can draw a line/plane/... between the different classes). For example, imagine you have two sets of data a and b : all values for set a are inside a circle located on the origine and all values for b are outside of this circle. If we add a third dimension that is the distance to the origine, we will obtain a cone pointing downward. All points from the set a are located below a certain height and all point from set b are above this height. We can thus split the set using a plane that cuts the cone at this height.

To apply SVM, we first filter all our data set with PCA:

$$\begin{aligned} S'_a &= S_a P \\ S'_b &= S_b P \\ S'_c &= S_c P \end{aligned}$$

We then feed the svm library³ (we could never have written this code) with two arrays : a list of labels and a list of vectors. We thus feed it with set S'_a , telling it these vectors have the label "a", S'_b with label "b" and S'_c with label "c". We then call the learn function.

To test an arbitrary movement \mathbf{t} , we must first move it into the vector space defined by PCA and then send it into SVM. The SVM black box will return the label for the test vector.

$$\begin{aligned} \mathbf{t}'_k &= \mathbf{t}' - \boldsymbol{\mu}'_k \\ &= \mathbf{t}P - \boldsymbol{\mu}'_k \end{aligned}$$

$$\text{label} = \text{svmlib.predict}(\mathbf{t}'_k)$$

³<http://www.csie.ntu.edu.tw/~cjlin/libsvm>

12. RUBY CODE

12.1. variance.

```

set = [1,4,3,6]
mu = set.inject(0) {|s,v| s + v } / set.size.to_f
sigma = set.inject(0) {|s,v| s + ((v - mu)**2) } / set.size.to_f

```

12.2. Mahalanobis distance.

```

class Array
  # transpose
  def t
    cols = []
    self[0].each_index do |i|
      cols[i] = map {|v| v[i]}
    end
    cols
  end

  def *(m)
    res = []
    each_index do |row|
      res[row] = []
      m[0].each_index do |col|
        res[row][col] = 0
        self[0].each_index do |i|
          res[row][col] += self[row][i] * m[i][col]
        end
      end
    end
    res
  end

  def det
    (self[0][0] * self[1][1]) - (self[1][0] * self[0][1])
  end

  def /(scalar)
    map {|row| row.map {|v| v / scalar}}
  end

  # vector subtraction
  def -(v)
    res = []
    self[0].each_index {|i| res[0][i] = self[0][i] - v[0][i]}
    res
  end
end

```

```

def to_s
  if self.size > 1 || self[0].size > 1
    if self.size == 1
      row_format = '[' + (" %3.1f" * self[0].size) + " ]\n"
    else
      row_format = '|' + (" %3.1f" * self[0].size) + " |\n"
    end
    self.inject("") {|s,row| s + sprintf(row_format, *row)}
  else
    self[0][0].to_s
  end
end

def map_index(&block)
  res = []
  self.each_index do |i|
    res[i] = yield(i,self[i])
  end
  res
end

set = [[1,1],
       [0,4],
       [2,1]]
# mean vector mu
mu = [[]]
set[0].each_index do |i|
  mu[0][i] = set.inject(0) {|s,v| s + v[i] } / set.size.to_f
end

# set - mu
s_mu = set.map do |v|
  r = []
  v.each_index do |i|
    r[i] = v[i] - mu[0][i]
  end
  r
end

# variance-covariance matrix
sigma = (s_mu.t * s_mu) / (set.size - 1.0).to_f

```

14

```
# inverse of sigma
if sigma.det == 0
  puts "Matrix not invertible"
  return
end

# invers of sigma (only for a 2x2 matrix)
sigma_inv = [[ sigma[1][1], -sigma[0][1] ],
             [ -sigma[1][0], sigma[0][0] ]] / sigma.det

puts "Sigma\n\n"
puts sigma.to_s
puts "\nSigma's inverse\n\n"
puts sigma_inv.to_s

puts "\n\nAll results express value^2 (before square root)"

[
  [[2 , 2 ]],
  [ [1 , 3 ]]],
  [[[1.5, 1.5]],
  [ [1.5, 2.5]]]
].each do |t1,t2|
  puts "\n-----"
  puts "t1   = #{t1}"
  puts "t2   = #{t2}"
  puts "t1-u = #{t1 - mu}"
  puts "t2-u = #{t2 - mu}\n\n"

  d1 = (t1 - mu).flatten.inject(0) {|s,v| s + v**2}
  d2 = (t2 - mu).flatten.inject(0) {|s,v| s + v**2}

  puts "d1   = #{d1}"
  puts "d2   = #{d2}\n\n"

  d1nE = (t1 - mu).flatten.map_index{|i,v| v**2 / sigma[i][i] }.inject(0) {|s,v| s + v}
  d2nE = (t2 - mu).flatten.map_index{|i,v| v**2 / sigma[i][i] }.inject(0) {|s,v| s + v}

  puts "d1nE = #{d1nE}"
  puts "d2nE = #{d2nE}\n\n"

  d1M = (t1 - mu) * sigma_inv * (t1 - mu).t
  d2M = (t2 - mu) * sigma_inv * (t2 - mu).t

  puts "d1M  = #{d1M}"
```

```
puts "d2M = #{d2M}\n\n"  
end
```