Dimension Reduction PCA.

Todays goal is to understand well what principle component ~~analysis~~ analysis does. We won't give a complete proof/derivation of PCA, but I will go deep enough to give you some intuition why and how the algorithm works.

Dimension Reduction

- What is it?
- why is it necessary.

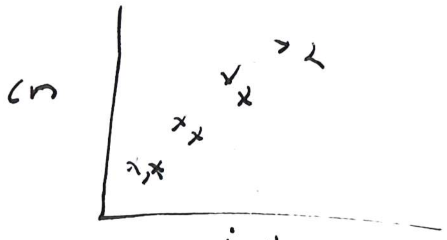
In short we want to transform high dimensional data into lower dimensional data

The most common transformations are Linear transformations, Why do we need it?

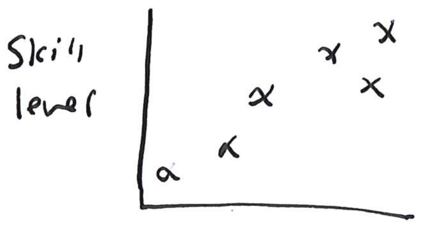
- There are two main reasons
  - Data redundancy
  - Improving the performance of learning algorithms

Data redundancy

Often it is possible that you have features that represent the same thing, or data that is highly correlated



Rounded to the nearest integer



Number of years experience

If you have a lot of features, then it might be difficult to keep track of the redundancy.

The curse of dimensionality

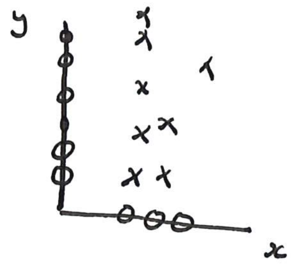
This is not a formal thing, but as the dimension increases then the complexity grows expontially. The higher the number of dimensions the more volume you have, hence your data becomes more sparse. You need much more data to train your learning algorithm to fix the weights/parameters (big problem with deep learning)

You would use dimension reduction so that your learning algorithm can learn something sensible.

You can also use dimension reduction for compression. If you can find a transformation that does not lose to much information, then you can compress that data.

What makes a good dimension reduction?

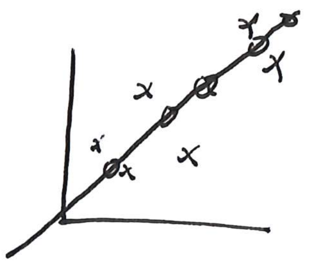
Let's think about the 2D - 1D case.



The circles are the predictions onto the x-axis or the y-axis.

It seems that the y-projection is better than the x-projection. We lose too much information with the x-projection because we clump everything together.

In general we are not projecting onto the axis but onto some d-dimensional hyperplane (line in the 2 to 1D case)



The optimisation problem that we need solve is that we want to find the best straight line (or d-dimensional hyperplane) that gives the least clumpiness, That is the points are the most spread out.

Very important

In what follows we will normalise our data. We will assume that the mean is always zero. Given N data points

$y_1, \ldots, y_n$ then

$$\bar{y} = \frac{1}{N} \sum_{i=1}^{n} y_i = 0$$

If your data is not normalised then it is easy to do by

$$y_i' = y_i - \bar{y}$$

It is always a good idea to normalise your data. It stops the learning algorithm from wasting time learning how your data is scaled. If one feature has very large values, then without scaling this feature will dominate.

Variance

We will use variance as a measure of how spread out the data is.

The variance is defined as

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

where $\bar{x}$ is the mean of the sample.

PCA is an algorithim that finds a projection that maximises the variance in each dimension.

I am going to follow the notation in the Rogers & Girolami book.

Linear Algebra

Input Space (M-dimension) $y \longrightarrow x$ Reduced dimension (D-dimension)

$$\begin{array}{ccc} \text{Input Space} & & \text{Reduced dimension} \\ (M\text{-dimension}) & & (D\text{-dimension}) \\ y & \longrightarrow & x \end{array}$$

Linear ~~Alts~~ tranformations can be express as matricy

$x = Wy$ W is a $D \times M$ matrix

$$\begin{array}{cc} & \longleftarrow M \longrightarrow \\ \updownarrow D & \begin{pmatrix} w_1 \text{———} \\ w_2 \\ \vdots \\ w_m \end{pmatrix} \end{array} \begin{pmatrix} y_1 \\ \vdots \\ y_M \end{pmatrix} = w_1 \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Another way of thinking about this is ~~tha~~

we need to find D vectors ~~[illegible]~~ $w_1, \ldots, w_d$

s.t. $x_i = w_i^T y$

We have to be a bit carefull.

All vectors will be column vectors so the above equation has the shape

$$\big(\qquad \big) \begin{pmatrix} \\ \\ \\ \end{pmatrix} = x_i.$$

So what should the PCA algorithm do?

We need to find D vector

- $w_1, \dots, w_D$ which are orthogonal

$i \neq j \Rightarrow w_i^T w_j = 0$. This is to avoid redundancy.

- For each component we want to maximize the variance.

If you are used to matrix algebra the derivation is actually quite simple. To make things understandable & to give you some intuition I will do the 2-D to 1-D case.

Given N data samples

$y_1, \dots, y_N$ if they are 2-D then

I will ~~[illegible]~~ denote the components as

$$y_i = \begin{pmatrix} y_{i_1} \\ y_{i_2} \end{pmatrix}$$

Again we are assuming that our data is normalised

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N}\begin{pmatrix} y_{i_1} \\ y_{i_2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Each point y_i gets transformed into a single real number x_i

$$x_i = w^T y_i = (w_1 \quad w_2)\begin{pmatrix} y_{i_1} \\ y_{i_2} \end{pmatrix}$$

We need to calculate the variance of the transformed data

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

What is the mean of our transformed data?

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} w^T y_i = w^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} y_i\right)}_{\text{equals zero because the data is normalised.}}$$

$$= w^T \bar{y} = \emptyset$$

In the 2-D case

$$\begin{pmatrix} y_{i_1} \\ y_{i_2} \end{pmatrix} \begin{pmatrix} y_{i_1} & y_{i_2} \end{pmatrix} = \begin{pmatrix} y_{i_1}^2 & y_{i_1} y_{i_2} \\ y_{i_1} y_{i_2} & y_{i_2}^2 \end{pmatrix}$$

It is just matrix multiplication Row x colums.

You can verify that $(w^T y_i)^2$ equals $w^T y_i y_i^T w$ in the 2-D case quite easily.

For the PCA algorithim we define a special matrix

~~$C = \frac{1}{N} \sum_{i=1}^{N} y_i^T y$~~

$$C = \frac{1}{N} \sum_{i=1}^{N} y_i y_i^T$$

If the data is not normalised then we use the matrix

$$C = \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y})(y_i - \bar{y})^T$$

So

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i)^2$$

We know that $x_i = w^T y_i$

So

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(w^T y_i)^2$$

$w^T y = y^T w$

$$= \frac{1}{N}\sum_{i=1}^{N} w^T y_i \cdot y_i^T w$$

$$= w^T\left(\frac{1}{N}\sum_{i=1}^{N} y_i y_i^T\right) w$$

What is going on with $y_i y_i^T$? What shape does it have?

$$\begin{array}{c}\uparrow\\ m\\ \downarrow\end{array}\begin{pmatrix} \\ y_i \\ \\ \end{pmatrix}\overset{\leftarrow\ m\ \rightarrow}{\left(\ \rule{2cm}{0.4pt}\ \right)} = \begin{array}{c}\uparrow\\ m\\ \downarrow\end{array}\overset{\leftarrow\ m\ \rightarrow}{\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}}$$

It is a $m \times m$ matrix

So our variance becomes

$$\sigma_x^2 = w^T C w$$

We need to maximize this, but it does not make sense to maximize the length of w. We want $w^T w = 1$.

So we want to maximize the expression

$$L = \underbrace{w^T C w}_{\text{Variance term}} - \lambda \underbrace{(w^T w - 1)}_{\text{Force } w^T w = 1.}$$

We can do this by looking at the partial derivative w.r.t. w

$$\frac{\partial L}{\partial w} = \left(\frac{\partial L}{\partial w_1} , \frac{\partial L}{\partial w_2} \right)^T$$

set $\frac{\partial L}{\partial w} = 0$ and solve.

There is a whole algebra of matrix derivatives and I am sorry I could not come up with a short proof. So you will have to believe me.

$$\frac{\partial w^T C w}{\partial w} = 2Cw \quad \text{and}$$ (See 13a for 2x2 case)

$$\frac{\partial \lambda (w^T w - 1)}{\partial w} = \lambda w$$

You need the fact that C is symmetric.

So $$\frac{\partial L}{\partial w} = 2Cw - 2\lambda w = 0$$

If we ignore the factor of 2 we are looking for solutions to the following equation

$$C.w = \lambda w$$

Such vectors have been well studied since the 18th century and are called eigenvectors

The matrix C, by definition is Symmetric
$C^T = C$.

For the 2x2 case we use $\frac{\partial}{\partial w_1}$ & $\frac{\partial}{\partial w_2}$

$$(w_1 \ \ w_2) \begin{pmatrix} c_1 & c_2 \\ c_2 & c_3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

$$= (w_1 \ \ w_2) \begin{pmatrix} c_1 w_1 + c_2 w_2 \\ c_2 w_1 + c_3 w_2 \end{pmatrix} = \begin{matrix} w_1 (c_1 w_1 + c_2 w_2) \\ + \\ w_2 (c_2 w_1 + c_3 w_2) \end{matrix}$$

x2

$$\frac{\partial}{\partial w_1} (c_1 w_1^2 + c_2 w_2 w_1 + c_2 w_1 w_2 + c_3 w_2^2$$

$$= \underline{2 c_1 w_1 + c_2 w_2 + c_2 w_2}$$ only works because C is Symmetric.

For the general case you can use indicies ~~[illegible]~~ madness or learn matrix calculus. for example

$$\frac{\partial w^T C w}{\partial w} = w^T (C + C^T)$$

You still have to be carefull about the shapes of the vectors & matrices

To find eigenvectors and values you

Solve the equation

$(C - \lambda I) w = 0$ Since ~~you~~ w is non-zero

you are looking for values of λ where the

determinate $\det(C - \lambda I) = 0$

All this boils down to is that PCA

is simply calculating the eigen-values & vectors of

C. There are lots of packages & methods

for doing this, which you should have covered in

Previous courses.

The higher the value of λ the higher the variance

$$\sigma_x^2 = w^T C w$$

$$\sigma^2 \underbrace{w^T w}_{=1} = w^T C w$$

$$\sigma^2 w = C w = \lambda w$$

Since C is real valued & Symmetric
$(C = C^T)$

the eigen~~v~~values & vectors have nice Property

- All the eigenvalues are real

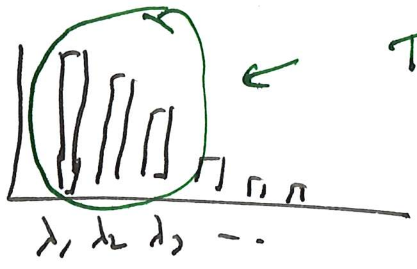
- There will be M eigenvalues

If this was not true then PCA would not be true..

To choose how may dimensions you reduce to look at the eigen-spectrum of C

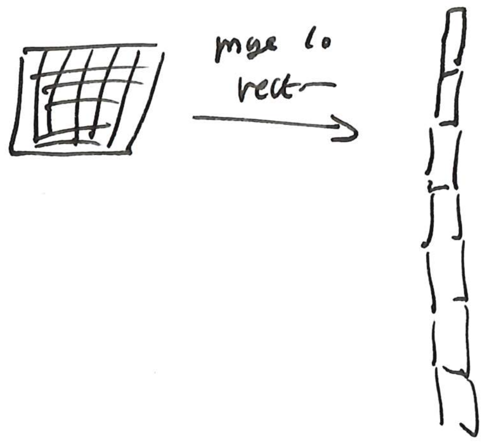
$$\underbrace{(w_1, \lambda_1), (w_2, \lambda_2), \ldots (w_k, \lambda_k)}_{\text{eigen vector value pair}}$$

s.t. $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots$



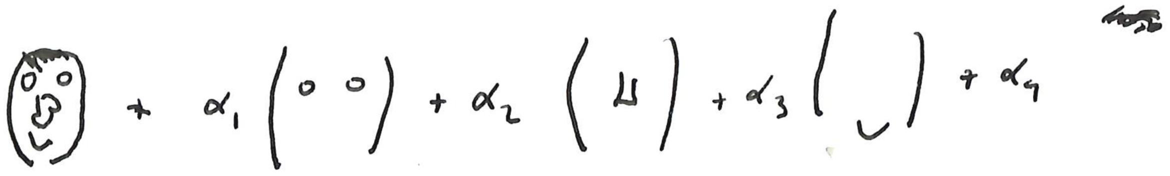
These are more important.

Applications Google ~~Eigenve~~ Eigenfaces



Do PCA

Learn common features



Vectors don't care about the 2D structure of the image, which is an advantage.