Machine Learning

Lecture 9

Principle Component Analysis

Justin Pearson[1]

mailto:it-1dl034@lists.uu.se

[1]http://user.it.uu.se/~justin/Hugo/courses/machinelearning/

Principle Component Analysis

The general idea of principle component analysis (PCA) is to reduce the number of dimensions in your data.

If you have some data and one feature is always a multiple of another feature this is useless information

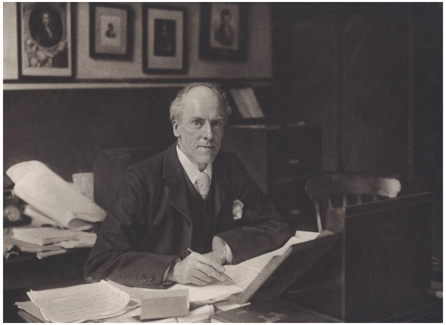
Reducing the number of dimensions helps your machine learning algorithms.

It is also a way of looking at features in your data.

Some of the maths today will get a bit heavy, but it is important to understand what is going on behind PCA. So that you can apply it.

Karl Pearson (As far I know, he is no relation) [2]

Principle component analysis was invented by Karl Pearson.

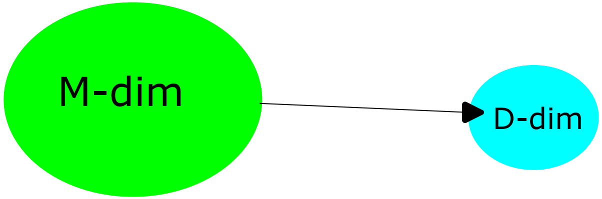


He invented much of modern statistics, and founded the first statistics department. By some accounts he wasn't a very nice person, and had a lot of views that we find abhorrent today. Statistics has a very murky history.

[2]Picture from Wikipedia. Taken the year PCA was invented.

Dimension Reduction General Idea

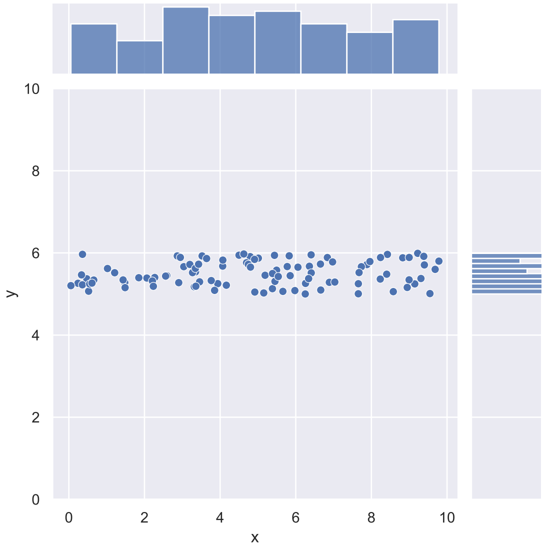
The general idea is to find some transformation from a higher-dimensional space, to a lower dimensional space.



The most usual transformation is a linear transformation.

But what is a good transformation?

Which axis gives the most information?



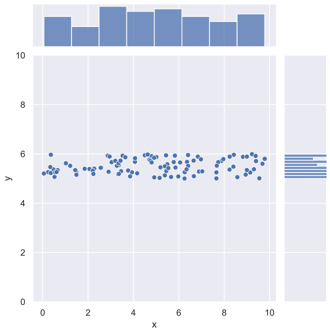
- The two histograms represent the projection onto the x and y axis with counting.
- The histograms on the x-axis counts the number, n, of points $(x, y_1), \ldots (x, y_n)$ with x as its coordinate.
- Similarly for the y-axis.

Interlude Seaborn is cool

```
import seaborn as sns
import numpy as np
import pandas as pd
number_of_data_points = 100
x = np.random.uniform(0,10,number_of_data_points)
y =  np.random.uniform(5,6,number_of_data_points)
df =
pd.DataFrame(data=np.column_stack((x,y)),
             columns = ["x","y"])
myplot = sns.jointplot(data=df, x="x", y="y")
myplot.ax_marg_y.set_ylim(0,10)
myplot.ax_marg_y.set_xlim(0,10)
```

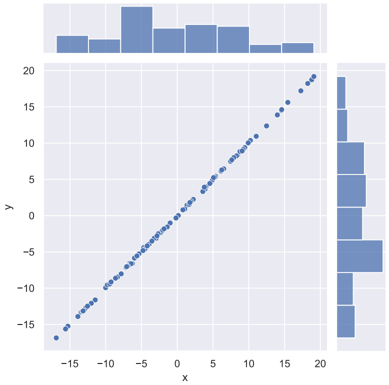
Seaborn sits on top of Matplotlib and lots of features for producing aesthetically pleasing figures that are also useful.

Which axis gives the most information?



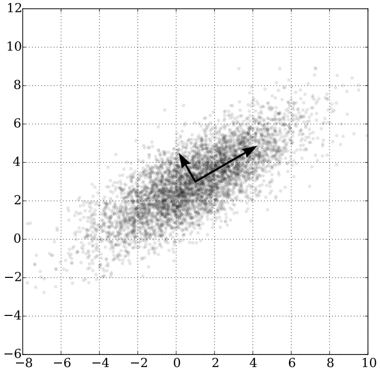
Obviously projecting on the y-axis would lose too much information.

Which axis gives the most information?



In this case it does not really matter which axis you project onto, but it is obvious that reducing this to one dimension will preserve most of the information in the original data set. In this case projecting onto the diagonal line $x = y$ is going reduce the amount of error.

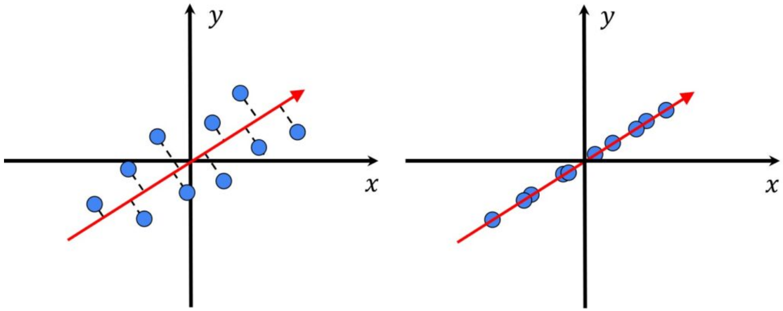
PCA is not about projecting to an axis[3]



In this case you will project onto the diagonal line.

[3]`https://commons.wikimedia.org/wiki/File:GaussianScatterPCA.svg`

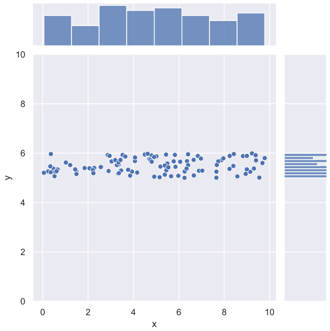
PCA is not about projecting to an axis[4]



You pick a lower dimensional hyperplane and project onto that.

[4]`https://datahacker.rs/principal-component-analysis-pca/`

What do we optimise?



The y-axis is bad because the data is too clumped together. So the optimising problem that we want to solve is find a linear projection that gives the least amount of clumping. To measure clumping we measure the standard deviation (or variance) of the data set, and find a projection that maximises the standard deviation for each value.

Notation

Things can get confusing. We are talking about i items of data, and m (or d) dimensional data. An m-dimensional vector has m components.

I will try to be consistent.

- The ith data item will be written as $x^{(i)}$.
- The jth component of a vector will the written x_j

This means that we will see things like:

$$x_j^{(i)}$$

Important: Normalise your data

In what follows we will assume that our data has mean 0.
It makes the maths easier, but it always a good idea to normalise your data when you are doing machine learning. If you don't normalise your data then scaling factors can skew your projections. If you measure one feature in kg, and another feature in milligrams, then it looks like the kg feature is more important.
Assume that you have N data points $y^{(1)}, \dots, y^{(N)}$, then the average is

$$\overline{y} = \frac{1}{N}\sum_{i=1}^{n} y^{(i)}$$

Then the normalised average data is

$$y'^{(i)} = y^{(i)} - \overline{y}$$

Of course scikit-learn provides functions to normalise your data.
It is often a good idea to normalise the variance as well.

Variance

Given N data points $x^{(1)}, \ldots, x^{(N)}$ where $\overline{x}$ is the average then the variance is

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

Principle Component Analysis is an algorithm that finds a projection that maximises the variance in each dimension.

Linear Algebra

General idea:

$$\begin{array}{ccc} \text{Input Space} & \longrightarrow & \text{Reduced Space} \\ \text{(m-dimensions)} & & \text{(d-dimensions)} \end{array}$$

We are going to find a linear transformation W.

$$y \in \mathbb{R}^m \xrightarrow{W} x \in \mathbb{R}^d$$

Linear Transformations as Matrices

$$x = Wy$$

Means that W is a $d \times m$ matrix.

$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & \cdots & \cdots & w_{2m} \\ \vdots & & & \vdots \\ w_{d1} & \cdots & \cdots & w_{dm} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

In what follows it is important to keep track of the shape of matrices and vectors.

Linear Algebra

Another way of thinking about linear mappings is a list of d vectors $w^{(1)}, \dots, w^{(d)}$ where

$$x_i = {w^{(i)}}^T y$$

If we represent vectors are columns then we need the transpose. The above equation has the shape:

$$\begin{pmatrix} w_{i1} & \cdots & w_{im} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = x_i$$

What should the PCA algorithm do?

Find d vectors $w_1, \ldots, w_d$ such that:

- For all $i \neq j$, ${w^{(i)}}^T w^{(j)} = 0$. That is they are orthogonal. This avoids are unnecessary redundancy. There are no spurious linear relationships.
- For each component x_i we maximise the variance.

$$\frac{1}{N}\sum_{j=1}^{N}(x_j^{(i)} - \overline{x_j})^2$$

- $\overline{x^i}$ is the average in dimension i.

The whole derivation is just matrix algebra.

PCA derivation

The whole derivation is just matrix algebra, but to make the slides understandable we'll do the special case of a 2-D to 1-D transformation.

Assume that you have N data samples with mean 0:

$$y^{(1)}, \dots, y^{(N)}$$

Each data sample is represented as a column vector:

$$y^{(i)} = \begin{pmatrix} y_1^{(i)} \\ y_2^{(i)} \end{pmatrix}$$

PCA derivation

So each vector $y^{(i)}$ gets transformed into a 1-dimensional point $x^{(i)}$. Our matrix W has shape 1×2. So we are just looking for a vector W.

$$x^{(i)} = w^T y^{(i)} = \begin{pmatrix} w_1 & w_2 \end{pmatrix} \begin{pmatrix} y_1^{(i)} \\ y_2^{(2)} \end{pmatrix}$$

We need to calculate the variance of the transformed data:

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

The mean of the transformed data

We just use our transformation:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{T} x^{(i)} = w^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} y^{(i)}\right)}_{\text{equals } 0}$$

So the transformed mean is 0 because the original data had mean 0.

The variance of the transformed data

We need to calculate:

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

Again using the transformation we get

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

The variance of the transformed data

So

$$(w^T y^{(i)})^2 = (w^T y^{(i)})(w^T y^{(i)})$$

and $w^T y^{(i)} = {y^{(i)}}^T w$ by general properties of the transpose.

$$(AB)^T = B^T A^T$$

This gives

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

Towards the covariance matrix

What shape does ${y^{(i)}}{y^{(i)}}^T$ have?

So $y^{(i)}$ is a 2×1 column vector and ${y^{(i)}}^T$ is a 1×2 row vector. So the product should be a 2×2 matrix.

$$\begin{pmatrix} y_1^{(i)} \\ y_2^{(i)} \end{pmatrix} \begin{pmatrix} y_1^{(i)} & y_2^{(i)} \end{pmatrix} = \begin{pmatrix} {y_1^{(i)}}^2 & y_1^{(i)} y_2^{(i)} \\ y_2^{(i)} y_1^{(i)} & {y_2^{(i)}}^2 \end{pmatrix}$$

This is just applying the rules of matrix multiplication. Rows by Columns.

Note that for example ${y_1^{(i)}}^2$ is $y_1^{(i)} y_1^{(i)}$.

Towards the covariance matrix

Define a special matrix :

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

So the variance becomes

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

The Covariance Matrix

For m-dimensional data that is not normalised you get an $m \times m$ matrix:

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

The covariance matrix is used quite a lot in statistics.

So in general the variance of the transformed data is going to be

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

Maximising the Variance

We need to maximise:

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

For various reasons it does not make sense to maximise the length of W. So we want $W^T W = 1$.

So we want to maximise the expression:

$$L = w^T C w - \underbrace{\lambda(w^T w - 1)}_{\text{forces } w^T w = 1}$$

Maximising the Variance

Look at the partial derivative with respect to W and solve.

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

Set

$$\frac{\partial L}{\partial w} = 0$$

and solve.

Of course to do this properly you have to show this gives the maximum, and not a minimum or a maximum.

Matrix Derivatives

There is a whole algebra of matrix derivatives, and the proofs are quite involved. You can either take my word for it, or do it for the 2×2 case.

$$\frac{\partial w^T C w}{\partial w} = 2Cw$$

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

So

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

Eigenvectors

Ignoring the constant factor 2 we are looking for solutions to the equation:

$$Cw = \lambda w$$

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

To find eigenvector and values you solve the equation:

$$(C - \lambda I)w = 0$$

Since W is going to be a non-zero vector you are looking for values of λ where the determinate

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

What is PCA doing?

All this boils down to is that PCA is simply calculating the eigenvalues and eigenvectors of the covariance matrix C. There are lots of packages and methods for doing this.

The higher the value of λ the higher the variance.

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

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

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

Nice properties of C

- C is real valued and symmetric:
$$C = C^T$$

This means that the eigenvalues and vectors have nice properties:

- All the eigenvalues will be real.
- There will be M eigenvalues.

If this was not true, then PCA would not work.

So calculate your covariance matrix C. Look at the set of eigenvector value pairs. sorted by λ.

$$(w_1, \lambda_1), (w_2, \lambda_2), \ldots, (w_m, \lambda_m)$$

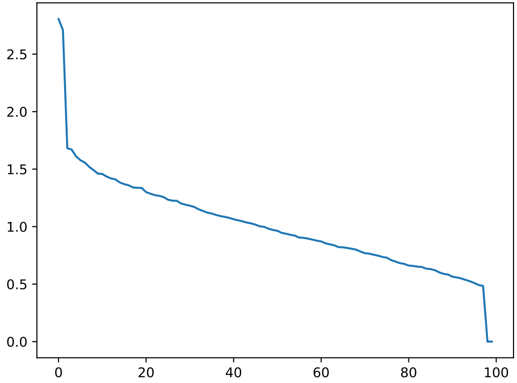
With $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$.

The eigenvalues start to drop dramatically , and you can use this as a guide to how many dimensions you will need.

You can use d of these m eigenvectors to give you a $d \times m$ matrix W.

Eigen-spectrum example

Start with a randomly generated 100 dimensional data set and we get the following graph:

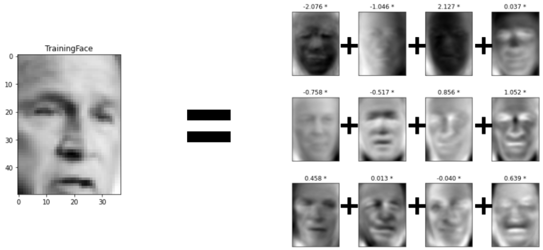


Code

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
X, y = make_classification(n_samples=1000,n_features=100)
n_samples = X.shape[0]
pca = PCA()
X_transformed = pca.fit_transform(X)
eigenvalues = pca.explained_variance_
print(eigenvalues)
plt.plot(eigenvalues)
```

Eigen-faces

An image just a list of numbers and can be treated as vector. You can then do PCA on that and you get something like[5]:



You can then use the PCA reduced data set as an input to some machine learning algorithm.

[5]Picture taken from
`https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/`

Principle Component Analysis

- Finds a linear transformation that reduces the dimension of the data set that loses the least amount of information.
- Information is preserved by maximising the variance in each dimension.
- PCA essentially computes eigenvalues and vectors of the covariance matrix. The eigenvalues tell you how important these features are.
- You combine d of the eigenvectors to get your linear transformation W.

It is a way of compressing data, and a way of removing features that are not important.