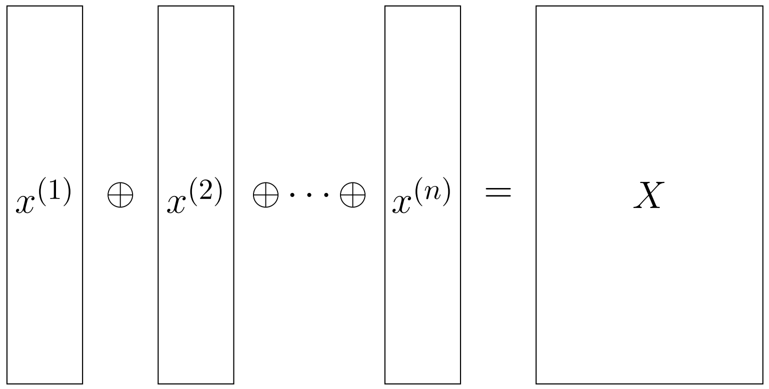
Chapter 2: Principal Component Analysis

John D Mangual

Machine learning is just linear regression and best fit lines. And that is just the Pythagorean theorem. Today let's discuss **Principal Component Analysis**.

Our data points are $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^n$ so we can merge them into a single matrix $X \in \mathbb{R}^{m \times n}$.



In Python is this done with the `np.hstack` or `np.vstack` commands (where `numpy` is shortened to `np`).

In beginning statistics we search for the best fit line. That means all our data points x approximately solve the same equation:

$$Ax + B \approx 0$$

What are the dimensions of A and B? Our vectors x have shape $1 \times n$ and the zero has shape 1×1, so that A has shape $n \times 1$ and B has shape 1×1. However, reduction to a line may lose much information.

What if we need more features? Let's take $n_0 < n$ feature.

What are the dimensions of A and B? Our vectors x have shape $1 \times n$ and the zero has shape $1 \times n_0$, so that A has shape $n \times n_0$ and B has shape $1 \times n_0$.

What is a reasonable numbe of data points? Some data sets brag as much as a billion, $m = 10^9$ data points. Each row of data could have as much as $n = 100$ features (or more) and we could like to reduce that to $n_0 = 10$.

If our data set included a few bit of inforation about each person:

- Height
- Favorite Ice Cream
- Birthday
- Occupation

Can we infer the **Birthday** and **Occupation** from the height and favorite ice cream flavor? Hopefully there is some type of correlation!

So we will take our data points X and find the covariance matrix. Hopefully our numbers do not have too much mistakes because they may propagage throughout our computation!

$$\text{Covariance} = X^T X \in \mathbb{R}^{m \times m}$$

What is the shape of our matrix? It is always okay to multiply a matrix with its own transpose:

$$(m \times n) \cdot (n \times m) = m \times m \quad \text{or} \quad (n \times m) \cdot (m \times n) = n \times n$$

n is the **number of features** and m is the **number of data points**, so we are doing the one on the right.

Next we decide which features are the most important. Using the eigenvalue decomposition:

$$X^T X \sim \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

Our matrix is not only square, but they are symmetric! So the eigenvalues $\lambda \in \mathbb{R}$ can be compared.

$$\lambda_1 > \lambda_2 > \cdots > \lambda_n$$

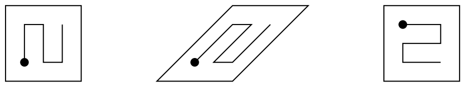
An example of a matrix whose eigenvalues cannot be a real number is a **rotation** by $90°$:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \sim \begin{bmatrix} \sqrt{-1} & 0 \\ 0 & \sqrt{-1} \end{bmatrix}$$

Here is a 2×2 matrix which is not diagonal. There is only one eigenvector:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

This is called a **shear**.



Our covariance matrix is symmetric: $(XX^T)^T = X^T(X^T)^T = XX^T$, so out our eigenvalues, our **features** are real numbers. In practice these features will be linear combinations and therefore totally ridiculous:

$$\frac{1}{2} \times \text{Height} + 2 \times \text{Birthday}$$

Let us proceed with our analysis of Principal Component Analysis.

The goal of PCA is to somehow "shrink" the data in to more mangeable form. This could be:

- shrinking an image file using JPEG, PNG, GIF or some other file format
- shrinking a WAV file to MP3
- shrinking a video to fit on your mobile phone

All of these compressions can be modeled as linear transforms from a space of $n \times 1$ vectors to smaller $n_0 \times 1$ vectors:

$$f : x \in \mathbb{R}^n \mapsto c \in \mathbb{R}^{n_0}$$

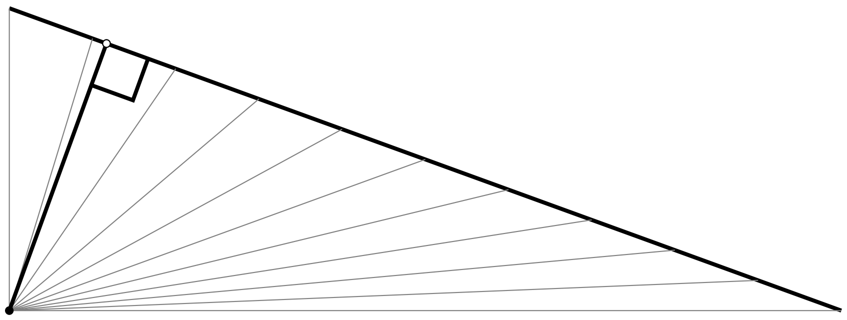
and this is what we call a **lossy** compression (or a "matrix"). How could it be possible to keep all this information? We simply cannot.

If we are lucky there will be some correlation and we can recover some of the information we have lost. $x = g(c)$ and $c = f(x)$ with $x \approx (g \circ f)(x)$

Our deep learning textbook has two simplifications that f and it's "pseudoinverse" g are both linear transformations:

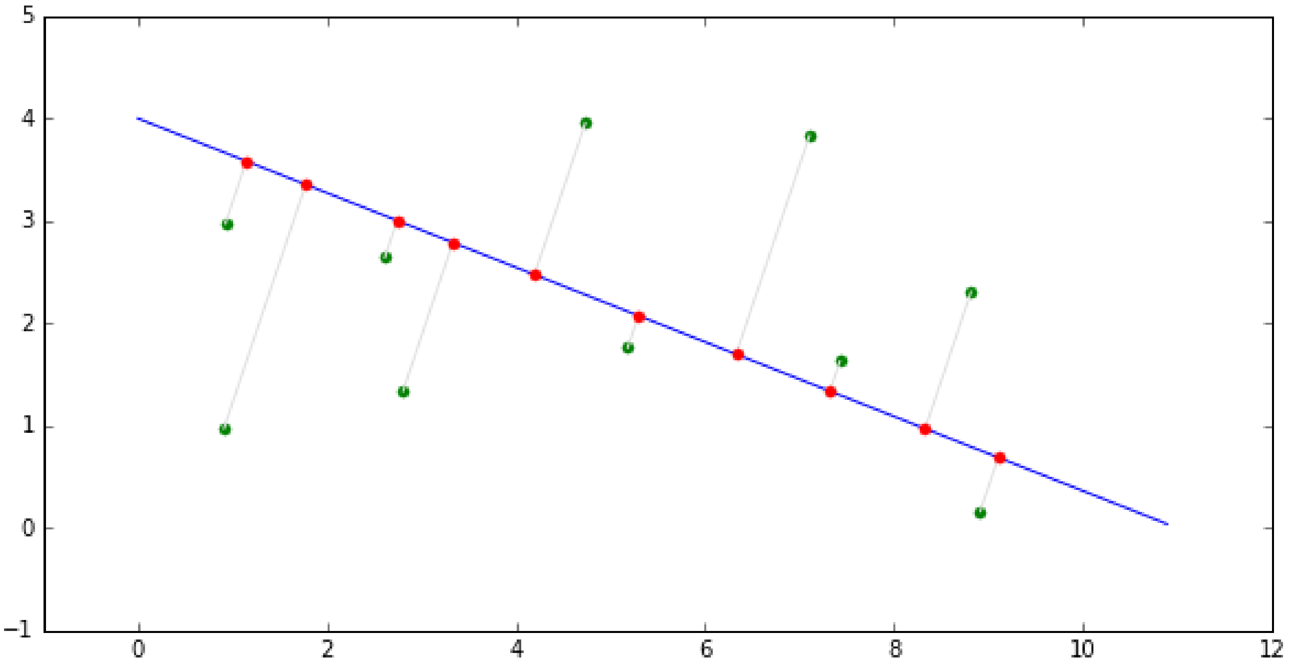
- $c = f(x) = D\ \ x$
- $x = g(c) = D^T x$
- $x = (g \circ f)(x)$

It will be impossible to have all these true for all our data points, so what is the best we can do?



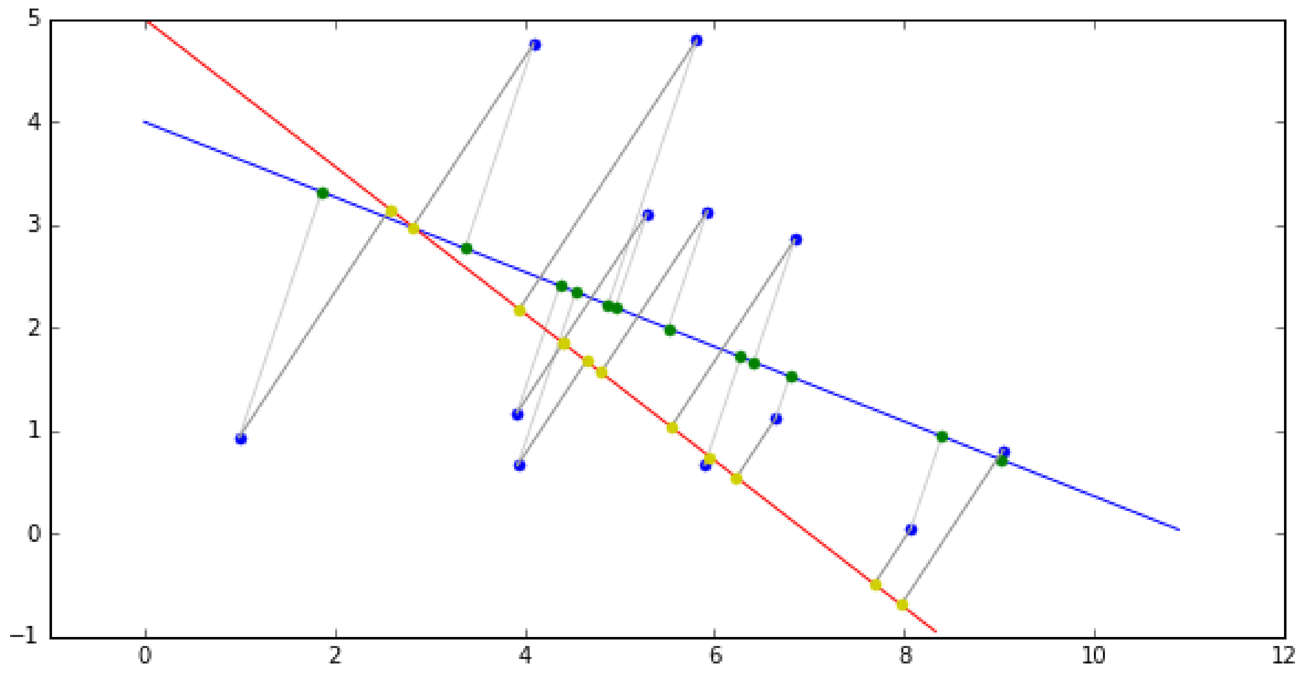
The closest point from one point to the line can be found by dropping a perpendicular. Even if a curve, the tangent line will be perpendicular.

If there are many points, then we have many points we can drop to a particular line. And we want the sum of the squares of all these distances.



We include some random points and a candidate best fit line.

Ng's Coursera course *agonizes* over best fit line, since he knows I am right.



This example is not the most aesthetic, just like real data is not.

Which one is a better fit?

Let's do some basic linear algebra (despite its simplicity it took a long time):

$$x \in \mathbb{R}^N \overset{f}{\mapsto} c = f(x) \in \mathbb{R}^M \overset{g}{\mapsto} x \approx (g \circ f)(x) \in \mathbb{R}^N$$

This looks just like inverse functions, but if $M \gg N$ there can be no inverse that works everywhere. This is why we need a **pseudoinverse**. I don't even know the definition.

Let's construct one a pseudoinverse by finding the constraints.

$$x \in \mathbb{R}^N \overset{f}{\mapsto} D\,x \in \mathbb{R}^M \overset{g}{\mapsto} x \approx D^T D\,x \in \mathbb{R}^N$$

and if x lies perfectly on the best-fit line (or best fit linear subspace)

$$D^T D = I_{M \times M}$$

This is a **rotation** (or "orthogonal matrix"). Except this D is an $N \times M$ matrix up to rotations, so this is an element of the Grassmanian.[1]

Why did we pick the transpose? We are trying to solve a minimization problem:

$$\min \big|\big|x - (g \circ f)(x)\big|\big|^2$$

If we decide the decoder function is $g(c) = D\,c$, we can try to solve this simplified problem.

$$\min_c \big|\big|x - D\,c\big|\big|^2$$

The closest we can get to an inverse is by the transpose, $c = f(x) = D^T x$.

Best Fit Hyperplane I am trying to find a way of encoding my length N with less information, just N bits and it should work reliably on all my existing data points. So I am going to guess that all my points live on an M dimensional subspace.

$$\mathrm{Gr}(N, M) = O(N)/\big(O(M) \times O(N - M)\big)$$

and D is an element of this Grassman space. And we would like to minimize the error obtained over all possible choices of D:

$$\min_D \big|\big|x - (D^T D)x\big|\big|^2 = \min_D \Big|\Big|x\left(I - (D^T D)\right)\Big|\Big|^2$$

To me this is just another L^2 norm. The Grassmanian is compact and there must be a minimum, found by a method of your choice.

[1] https://en.wikipedia.org/wiki/Grassmannian

Then we can prove the minimizing D is found by in fact the covariance of our data points $X^T X$. And we can prove that any flow on the Grassmanian (such as **stochastic gradient descent**)

If I google the phrase "principal component analysis grassmanian" more than zero things arise. Including this one:

- Matthew Nokleby, Miguel Rodrigues, Robert Calderbank **Discrimination on the Grassmann Manifold: Fundamental Limits of Subspace Classifiers** `arXiv:1404.5187`

I don't want to be stuck analyzing giant spreadsheets all day, when we all have things we want to do. At least someone has observed the geometric approach to PCA, that I am advocating.