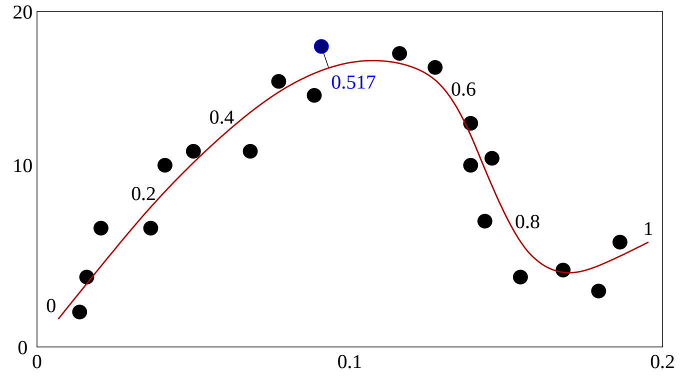
Dimensionality Reduction

High dimensional data is often “really” lower-dimensional: For example:



These points all lie near a curve. Perhaps all that matters is where the points lie on this curve, with the small departures from the curve being unimportant.

If so, we can reduce this 2D data to one dimension, by just projecting each point to the nearest point on the curve. Specifying a point on the curve requires just one coordinate. For example, the blue point at $(0.9, 18)$ is replaced by 0.517.

Manifolds and Embedding

In general, the p-dimensional data points might lie near some m-dimensional surface, or *manifold*.

Points in an m-dimensional manifold can (in each local region) be specified by m coordinates. Eg, points on a sphere can be described by “latitude” and “longitute” coordinates, so the sphere is a 2D manifold.

An p-dimensional “embedding” of the manifold is a map from points on the manifold to p-dimensional space.

Finding an embedding of a lower-dimensional manifold that the data points lie near is one form of unsupervised learning. We’d like to be able to map each point to the coordinates of the point closest to it on the manifold.

Some methods don’t really find a manifold and an embedding — they just assign m coordinates to each p-dimensional training case, but don’t have any way of assigning low-dimensional coordinates to new test cases. Such methods may still be useful for visualizing the data.

Hyperplanes

In the simplest form of dimensionality reduction, the manifold is just a hyperplane.

An m-dimensional hyperplane through the origin can be specified by a set of m basis vectors in p-dimensional space, which are most conveniently chosen to be orthogonal and of unit length.

If $e_1, \ldots, e_m$ are such a basis, the point in the hyperplane that is closest to some p-dimensional data point x is the one with the following coordinates (in terms of the basis vectors):

$$e_1^T x, \ldots, e_m^T x$$

If we want a hyperplane that doesn’t go through the origin, we can just translate the data so that this hyperplane does go through the origin.

Principal Component Analysis

Principal Component Analysis (PCA) is one way of finding a hyperplane that is suitable for reducing dimensionality.

With PCA, the first basis vector, e_1, points in the direction in which the data has maximum variance. In other words, the projections of the data points on e_1, given by $e_1^T x^{(1)}, \ldots, e_1^T x^{(n)}$, have the largest sample variance possible, for choice of unit vector for e_1.

The second basis vector, e_2, points in the direction of maximum variance subject to the constraint that e_2 be orthogonal to e_1 (ie, $e_2^T e_1 = 0$).

In general, the i’th basis vector, also called the i’th principal component, is the direction of maximum variance that is orthogonal to the previous $i-1$ principal components.

There are p principal components in all. Using all of them would just define a new coordinate system for the original space. But if we use just the first m, we can reduce dimensionality. If the variances associated with the remaining principal components are small, the data points will be close to the hyperplane defined by the first m principal components.

Finding Principal Components

To find the principal component directions, we first centre the data — subtracting the sample mean from each variable. (We might also divide each variable by its sample standard deviation, to eliminate the effect of arbitrary choices of units.) We put the values of the variables in all training cases in the $n \times p$ matrix X.

We can now express p-dimensional vectors, v, in terms of the eigenvectors, $e_1, \ldots, e_p$, of the $p \times p$ matrix X^TX. Recall that these eigenvectors will form an orthogonal basis, and the e_k can be chosen to be unit vectors. I'll assume they're ordered by decreasing eigenvalue. We'll write $v = s_1e_1 + \cdots + s_pe_p$.

If v is a unit vector, the projection of a data vector, x, on the direction it defines will be Xv. The sample variance of the training data projected on this direction is

$$\begin{aligned}(1/n)(Xv)^T(Xv) &= (1/n)v^T(X^TX)v \;=\; (1/n)v^T(s_1\lambda_1e_1 + \cdots s_p\lambda_pe_p) \\ &= (1/n)(s_1^2\lambda_1 + \cdots s_p^2\lambda_p)\end{aligned}$$

where λ_k is the eigenvalue associated with the eigenvector e_k, and $\lambda_1 \geq \cdots \geq \lambda_p$.

To maximize this variance we should set $s_1 = 1$ and other $s_j = 0$, so that $v = e_1$. Among unit vectors orthogonal to e_1, the one maximizing the variance is e_2, etc.

More on Finding Principal Components

So we see that we can find principal components by computing the eigenvectors of the $p \times p$ matrix X^TX, where the $n \times p$ matrix X contains the (centred) values for the p variables in the n training cases (in other words, $X_{ij} = x_j^{(i)}$). We choose eigenvectors that are unit vectors, of course. The signs are arbitrary.

Computing these eigenvectors takes time proportional to p^3.

What if p is big, at least as big as n? Eg, gene expression data from DNA microarrays often has $n \approx 100$ and $p \approx 10000$. Then X^TX is singular, with $p - n + 1$ zero eigenvalues. There are only $n - 1$ principal components, not p.

We can find the $n - 1$ eigenvectors of X^TX with non-zero eigenvalues from the eigenvalues of the $n \times n$ matrix XX^T, in time proportional to $n^3 + n^2p$. If v is an eigenvector of XX^T with eigenvalue λ, then X^Tv is an eigenvector of X^TX, with the same eigenvalue:

$$(X^TX)(X^Tv) \;=\; X^T(XX^T)v \;=\; X^T\lambda v \;=\; \lambda(X^Tv)$$

So PCA is feasible as long as *either* of p or n is no more than a few thousand.

What is PCA Good For?

Seen as an unsupervised learning method, the results of PCA might be used just to gain insight into the data.

For example, we might find the first two principal components, and then produce a 2D plot of the data. We might see interesting structure, such as clusters.

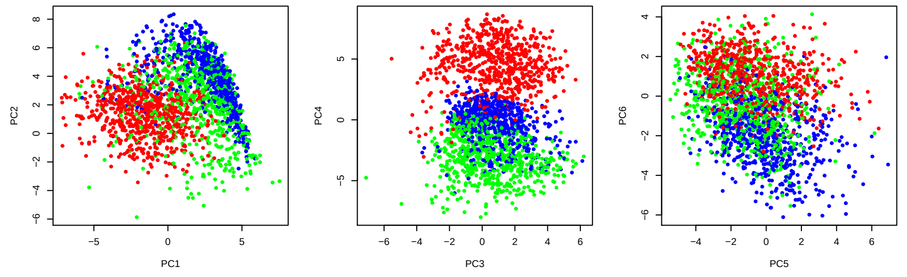
PCA is also used as a preliminary to supervised learning. Rather than use the original p inputs to try to predict y, we instead use the projections of these inputs on the first m principal components. This may help avoid overf itting. It certainly reduces computation time.

There is no guarantee that this will work — it could be that it is the small *departures* of x from the m dimensional hyperplane defined by the principal components that are important for predicting y.

Example: Zip Code Recognition

I tried finding principal componenents for data on handwritten zip codes (US postal codes). The inputs are pixel values for an 8×8 image of the digit, so there are 256 inputs. There are 7291 training cases.

Here are plots of 1st versus 2nd, 3rd versus 4th, and 5th versus 6th principal components for training cases of digits "3" (red), "4" (green), and "9" (blue):



Clearly, these reduced variables contain a lot of information about the identity of the digit — probably much more than we'd get from any six of the original inputs.

Pictures of What the Principal Components Mean

Directions of principal components in input space are specified by 256-dimensional unit vectors. We can visualize them as 16×16 "images". Here are the first ten: