Principal Component Analysis (PCA): A Pedestrian Review

Wagner L. Truppel

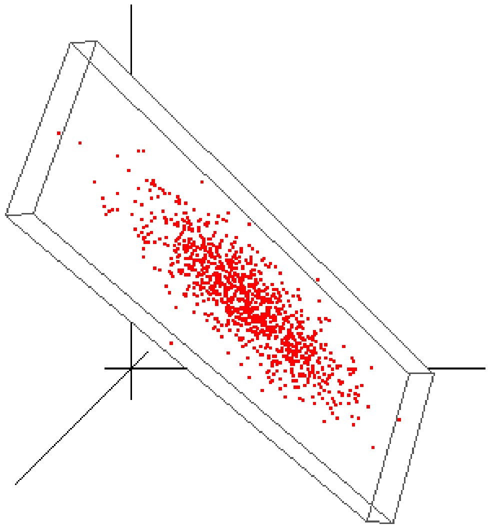
October 17, 2000

Suppose you have a dataset $\mathcal{S}$ consisting of N data points, each data point being an ordered sequence of d measurements of some kind. A natural representation is to think of each data point as a (column) vector in a space of d dimensions:

$$\begin{aligned} \mathcal{S} &= \{\mathbf{x}_n\}, \quad n = 1, 2, \ldots, N\,, \\ \mathbf{x}_n &= (x_{n1}, x_{n2}, \ldots, x_{nd})^\top\,. \end{aligned}$$

(The symbol $^\top$ represents the operation of *transposition*, which turns rows into columns and columns into rows. $(x_{n1}, x_{n2}, \ldots, x_{nd})$ is referred to as a *row* vector.)

In $d = 3$ dimensions, the dataset can be visualized as a scatter plot, which might look like, for example, the plot shown below.



This particular dataset strikes us as being very 'flat' and suggests the possibility that perhaps the variability along the 'thin' direction is due simply to random noise and that the only meaningful variability is contained in a subspace, in this case, of dimension 2. Or, perhaps, all that we really want or need to do is to store the data in a way that preserves most of its features without having to store *all* of it. This is often the case, for instance, when the data consists of large images.

Visualizing the entire dataset in a single plot when $d > 3$ is not feasible, so the question arises as to whether there is a way to automate this process of 'throwing away' those dimensions that seem irrelevant. This is what *Principal Component Analysis* (PCA) lets us accomplish in a principled way.

Let's try to understand what PCA does, from a geometric point of view. First, notice that our goal is to determine the directions in space along which the data presents the most variability. Those are the 'important' directions, the ones we want to keep. Now, data variability is measured by the dataset's *sample covariance matrix* $\mathbf{\Sigma}$, defined as[1]

$$\mathbf{\Sigma} \equiv \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \mu) \cdot (\mathbf{x}_n - \mu)^\top \,, \text{ where } \mu \equiv \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \text{ is the dataset's } \textit{sample mean.}$$

What is the geometrical interpretation of this matrix? In order to answer this question, let's look at the *quadratic form*

$$\Phi(\mathbf{x}) = (\mathbf{x} - \mu)^\top \cdot \mathbf{\Sigma} \cdot (\mathbf{x} - \mu) \,,$$

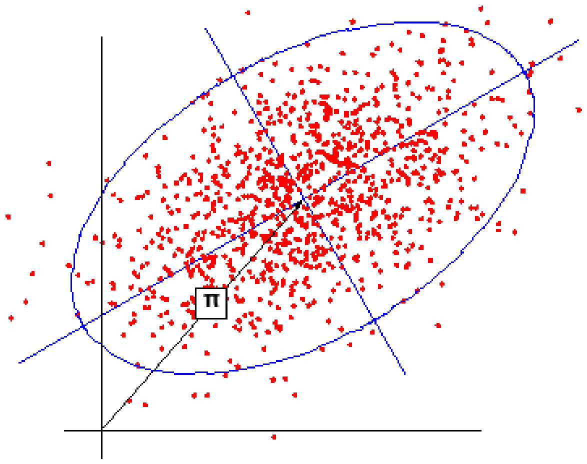
as a function of an arbitrary vector $\mathbf{x}$. Note that $\Phi(\mathbf{x})$ is an ordinary real number, not a vector or a matrix (the $\cdot$ represents the usual matrix multiplication). In addition, being in some sense proportional to $\mathbf{\Sigma}$, $\Phi(\mathbf{x})$ is also a measure of variance, of how much $\mathbf{x}$ differs from the mean μ.

Now, for any number of dimensions (any d), the locus of all vectors $\mathbf{x}$ such that $\Phi(\mathbf{x})$ has a fixed value, say $\Phi(\mathbf{x}) = \Phi_0$, is generally a *hyper-ellipsoid.* This is just a fancy word to express the idea that, when $d = 3$, the locus is the surface of an ellipsoid and, when $d = 2$, it is the curve traced out by an ellipse. The figure below gives an example in 2 dimensions.

It is apparent from the figure that the directions of largest variance are the so-called *principal axes* of the hyper-ellipsoid. Thus, it seems reasonable that to find these directions of largest variance, all we need to do is *diagonalize* the sample covariance matrix. In other words, we need to rotate the axes around their common origin (the mean μ) until the new axes point along the directions of maximum variance.

To summarize, geometrically, PCA is nothing more than a translation of the center of our original set of axes (which the original data points are referred to) to the dataset sample mean, followed by a rotation around this new origin. The rotation is chosen so as to maximize the lengths of the axes which are contained within the hyper-ellipsoid. Once we have the new set of axes, the sample covariance

[1]Sometimes, $\mathbf{\Sigma}$ is defined with a factor $1/(N-1)$ in front instead of $1/N$, as shown. This distinction does not affect the concepts being discussed here.



matrix's principal axes, we can write any data point (a vector of dimension d) as a linear combination of unit vectors (each also a vector of dimension d) pointing along those directions. By dropping the directions of least variance, we in effect produce a representation of the dataset in which the 'new' data points $\{\mathbf{y}_n\}$ have a smaller number of components.

Ok, how does this all work, mathematically? What's the algorithm to compute the directions of maximum variance?

We want to obtain a set of directions $\mathbf{u}_k$, $k = 1, 2, \ldots, d$, each one a unit vector of dimension d, such that along each of these directions the dataset presents the largest variance from the mean. Well, the departure of a single data point $\mathbf{x}_n$ from the mean μ is $(\mathbf{x}_n - \mu)$ and that departure, along the direction represented by the unit vector $\mathbf{u}_k$, is just the projection of $(\mathbf{x}_n - \mu)$ onto $\mathbf{u}_k$, namely, the dot product $\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)$. The variance of that one data point from the mean, along the direction of $\mathbf{u}_k$, is then $[\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)]^2$, and the total variance across the entire dataset reads

$$s_k = \sum_{n=1}^{N} [\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)]^2 = \sum_{n=1}^{N} [\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)][\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)]\,.$$

The goal is to maximize this quantity for each choice of k. But there are two ways to maximize s_k. We want to maximize it over the choice of possible *directions* for $\mathbf{u}_k$, not over the choice of their *magnitudes*. In fact, we already agreed that each $\mathbf{u}_k$ is a *unit* vector. Somehow we need to take into account the constraint that $\mathbf{u}_k$ has unit magnitude. This is done using the method of *Lagrange multipliers*. It basically amounts to adding *zero* to the quantity we want to maximize (so that we don't change that quantity), but zero written in a special way. We then proceed with the maximization as if the constraint was not a problem. The price to pay for this trick is that we now have an additional unknown quantity to determine, the Lagrange multiplier itself. So we redefine s_k as

$$s_k = \sum_{n=1}^{N} [\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)][\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)] + \lambda'_k \, (1 - \mathbf{u}_k^\top \cdot \mathbf{u}_k).$$

Note that since $\mathbf{u}_k$ has unit magnitude, the added term is indeed equal to zero. λ'_k is the Lagrange multiplier. We can now rewrite s_k as follows:

$$\begin{aligned} s_k &= \sum_{n=1}^{N} \left\{ \left(\sum_{i=1}^{d} u_{ki} \, (x_{ni} - \mu_i) \right) \left(\sum_{j=1}^{d} u_{kj} \, (x_{nj} - \mu_j) \right) \right\} + \lambda'_k \left[1 - \sum_{i=1}^{d} (u_{ki})^2 \right] \\ &= \sum_{i=1}^{d} \sum_{j=1}^{d} u_{ki} \, u_{kj} \sum_{n=1}^{N} (x_{ni} - \mu_i) \, (x_{nj} - \mu_j) + \lambda'_k \left[1 - \sum_{i=1}^{d} (u_{ki})^2 \right] \\ &= N \sum_{i=1}^{d} \sum_{j=1}^{d} \Sigma_{ij} \, u_{ki} \, u_{kj} + \lambda'_k \left[1 - \sum_{i=1}^{d} (u_{ki})^2 \right]. \end{aligned}$$

Maximization of s_k requires its derivative with respect to each component of $\mathbf{u}_k$ to vanish. Now,

$$\frac{\partial s_k}{\partial u_{k\ell}} = N \sum_{i=1}^{d} \sum_{j=1}^{d} \Sigma_{ij} \left(\frac{\partial u_{ki}}{\partial u_{k\ell}} \, u_{kj} + u_{ki} \, \frac{\partial u_{kj}}{\partial u_{k\ell}} \right) - 2 \, \lambda'_k \sum_{i=1}^{d} u_{ki} \, \frac{\partial u_{ki}}{\partial u_{k\ell}} \, .$$

Note that the first partial derivative is zero unless $i = \ell$, in which case the derivative is 1, and similarly for the second derivative. Thus,

$$\frac{\partial s_k}{\partial u_{k\ell}} = N \sum_{j=1}^{d} \Sigma_{\ell j} \, u_{kj} + N \sum_{i=1}^{d} \Sigma_{i\ell} \, u_{ki} - 2 \, \lambda'_k \, u_{k\ell} \, .$$

Since the covariance matrix is symmetric, however, the second term equals the first, and we find that imposing $\partial s_k / \partial u_{k\ell} = 0$ implies

$$\sum_{j=1}^{d} \Sigma_{\ell j} \, u_{kj} = \frac{\lambda'_k}{N} \, u_{k\ell} \equiv \lambda_k \, u_{k\ell} \quad \text{or, in matrix terms,} \quad \mathbf{\Sigma} \cdot \mathbf{u}_k = \lambda_k \, \mathbf{u}_k \, .$$

The last equation above embodies the mathematical prescription to compute the *eigenvalues* and *eigenvectors* of the matrix $\mathbf{\Sigma}$, a process also known as *matrix diagonalization*; λ_k is the eigenvalue associated with the eigenvector $\mathbf{u}_k$. As the index suggests, there are d pairs $\{\lambda_k, \mathbf{u}_k\}$. The eigenvalues don't all have to be distinct, but it's a property of matrix diagonalization that *all* eigenvalues of a *real and symmetric* matrix are *real* while the corresponding eigenvectors are *orthonormal* (they have unit magnitude and are pairwise orthogonal).

Actually, to be more precise, the last equation is the *definition* of matrix diagonalization, but not really a prescription of how to compute the eigenvalues and eigenvectors. One way to compute them, although not the most efficient, is to notice that the last equation above implies

$$(\mathbf{\Sigma} - \lambda_k \mathbf{I}) \cdot \mathbf{u}_k = \mathbf{0}$$

where $\mathbf{I}$ and $\mathbf{0}$ are the *identity* and the *zero* matrices, respectively. If we think of this as a system of linear equations on the coefficients of $\mathbf{u}_k$, then we know that there can only be a non-zero solution if the determinant of the matrix multiplying $\mathbf{u}_k$ vanishes. This gives us the matrix *characteristic polynomial*,

$$\det (\mathbf{\Sigma} - \lambda_k \mathbf{I}) = 0\,,$$

a polynomial in λ_k of degree equal to the size of the matrix (d, in our case). Solving this polynomial for its roots gives us the eigenvalues. Once in possession of the eigenvalues, we can use $(\mathbf{\Sigma} - \lambda_k \mathbf{I}) \cdot \mathbf{u}_k = \mathbf{0}$ to solve for each $\mathbf{u}_k$, given its associated eigenvalue.

How efficient is this, computationally? In the most general case, where $\mathbf{\Sigma}$ is not particularly sparse, direct diagonalization is fairly expensive, with a computational complexity of $\mathcal{O}(d^3)$ operations. More efficient methods exist, however, when all that one needs is the first few largest eigenvalues and their associated eigenvectors.

Once we have obtained the set $\{\lambda_k, \mathbf{u}_k\}$, we can proceed to create a compact representation of the original dataset, as follows. Recall that diagonalization of the sample covariance matrix amounts to a rotation around the sample mean vector. Hence, any data point $\mathbf{x}_n$ can be written as

$$\mathbf{x}_n = \mu + \sum_{k=1}^{d} \alpha_{nk} \mathbf{u}_k\,,$$

where α_{nk} is some real number, the 'amount' of $(\mathbf{x}_n - \mu)$ along the direction of $\mathbf{u}_k$. Since the eigenvectors are pairwise orthonormal, it turns out to be easy to solve the previous equation for each α_{nk},

$$\alpha_{nk} = \mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)\,.$$

So far, this is exact. In other words, knowing the set $\{\alpha_{nk}\}$, for a given n, is perfectly equivalent to knowing the data point $\mathbf{x}_n$ itself. Thus, the set $\{\alpha_{nk}\}$ is as good a representation of $\mathbf{x}_n$ as $\mathbf{x}_n$ itself is. However, if we were to keep only the largest eigenvalues, say, $d' < d$ of them, then we could associate to each data point $\mathbf{x}_n$ a 'new' data point $\mathbf{y}_n$, given by

$$\mathbf{y}_n = \mu + \sum_{k=1}^{d'} \alpha_{nk} \mathbf{u}_k\,,$$

which is represented by a *smaller* set of α values. Note that, in any case, we have a linear mapping from the space of data points $\mathbf{x}_n$ onto the space of the 'new' data points $\mathbf{y}_n$, since we can write

$$(\mathbf{y}_n - \mu) = \sum_{k=1}^{d'} \alpha_{nk} \mathbf{u}_k = \sum_{k=1}^{d'} [\mathbf{u}_k^\top \cdot (\mathbf{x}_n - \mu)]\, \mathbf{u}_k = \Big(\sum_{k=1}^{d'} \mathbf{u}_k \cdot \mathbf{u}_k^\top \Big) \cdot (\mathbf{x}_n - \mu) = \mathbf{T} \cdot (\mathbf{x}_n - \mu)\,,$$

where $\mathbf{T} \equiv \sum_k \mathbf{u}_k \cdot \mathbf{u}_k^\top$. Of course, if we're replacing $\mathbf{x}_n$ with $\mathbf{y}_n$, we must be incurring in some error. In other words, our 'compression' scheme is not perfect. The *sum of squared reconstruction errors* accumulated across the entire dataset is found to be

$$\begin{aligned} \mathrm{SSE}(d') &= \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{y}_n)^\top \cdot (\mathbf{x}_n - \mathbf{y}_n) = \sum_{n=1}^{N} \left(\sum_{k=d'+1}^{d} \alpha_{nk} \mathbf{u}_k^\top \right) \cdot \left(\sum_{\ell=d'+1}^{d} \alpha_{n\ell} \mathbf{u}_\ell \right) \\ &= \sum_{k=d'+1}^{d} \sum_{\ell=d'+1}^{d} \left(\sum_{n=1}^{N} \alpha_{nk}\, \alpha_{n\ell} \right) (\mathbf{u}_k^\top \cdot \mathbf{u}_\ell) = \sum_{n=1}^{N} \sum_{k=d'+1}^{d} (\alpha_{nk})^2 , \end{aligned}$$

where in the last step we used the fact that the eigenvectors have unit magnitude and are pairwise orthogonal to explicitly evaluate one of the sums.

It's easy to show, however, that this accumulated error achieves its smallest possible value when the α_{nk}'s are given by the PCA values deduced above.[2] This is an important property of PCA, worth rephrasing: *of all the possible linear mappings* $(\mathbf{y}_n - \mu) = \mathbf{T} \cdot (\mathbf{x}_n - \mu)$*, the choice of* $\mathbf{T}$ *prescribed by PCA is that which minimizes the sum of squared errors across the dataset.*

Let's finish by listing the properties, advantages, and drawbacks of PCA.

Properties and advantages of PCA:

- It has a simple and intuitive geometrical interpretation;
- It has a solid and well-known mathematical background;
- It represents the data in terms of directions in data space along which the data presents the most variance;
- It allows for a compact representation of the data (*data compression*);
- Data compression is *optimal*, in the sense of minimizing the sum of squared reconstruction errors across the dataset;
- Data compression ($\mathbf{x}_n \to \{\alpha_{nk}\}$) and data reconstruction ($\{\alpha_{nk}\} \to \mathbf{y}_n$) involve only simple and inexpensive operations once the required parameters (μ, $\mathbf{\Sigma}$, and $\{\lambda_k, \mathbf{u}_k\}$) have been computed.

Drawbacks of PCA:

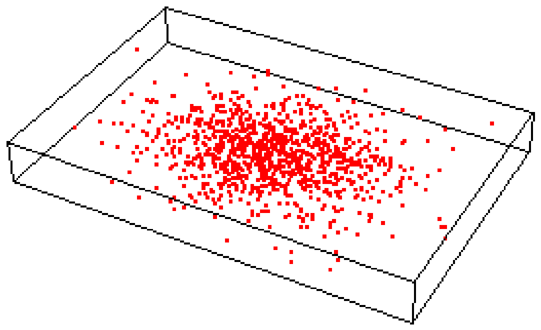
- Computation of the required parameters is expensive in spaces of high dimensionality. Computing $\mathbf{\Sigma}$ takes $\mathcal{O}(Nd^2)$ operations and its diagonalization requires $\mathcal{O}(d^3)$ operations;

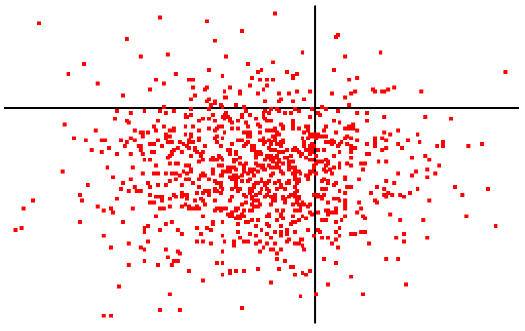
[2] Just compare the expressions for SSE and s_k and notice that the minimization of SSE proceeds in the exact same fashion as the maximization of s_k, with the same results in both cases.

- The number N of data points required to accurately sample the variance in the data and compute the sample covariance matrix $\mathbf{\Sigma}$ increases with the dimensionality d of the space. This is known as the *curse of dimensionality*: the higher the dimension of the data space, the more data points are needed for an accurate representation of that data;
- PCA has trouble dealing with missing information: *all* data points must be vectors of the same dimension;
- PCA does not define a proper probability distribution of its parameters. In essence, PCA is a static model, providing an all–or–nothing representation of the data. There is no notion of likelihood of the data given the model's parameters, and the only measure of model quality is the sum of squared reconstruction errors.

Examples:

Consider the 3-dimensional datasets shown below. Their PCA decompositions (where λ_k has already been divided by N) are also shown, as are the resulting reconstructed data points.





■