1 Principal Component Analysis

Principal component analysis is ubiquitous tool. In these notes we aim to demonstrate how it might be useful with an example from neuroscience.

Say we have a number of multidimensional data points, whose underlying joint probability density is some complicated function in high dimensions. Specifically, each data point is a $\mathbf{x}_i \in \mathbb{R}^n$. PCA consists of fitting a n-dimensional ellipsoid to the collection of data points. From the principal axes of the ellipsoid — natural analogues of the principal axes for the two dimensional ellipse — we can assess the directions in $\mathbb{R}^n$ along which the variance of our dataset is large or small. PCA is therefore a dimensionality reduction technique — it aims to discover the directions along which the most variation occurs in a dataset.

First we center our dataset $\{\mathbf{x}_i\}$ by subtracting the mean $\mathrm{E}[\mathbf{x}_i]$ from each $\mathbf{x}_i$. We then calculate the covariance matrix of the centered data, and diagonalize the covariance matrix. Its eigenvectors must be all mutually orthogonal because the covariance matrix is symmetric. We normalize these eigenvectors to get our principal axes, where the eigenvalue along that direction is proportional to the fraction of variance accounted for by this principal direction.

If our data to begin with were multivariate normally distributed, then this amounts to changing into a basis where our data are multivariate gaussian but with a *diagonal* covariance matrix, meaning that different directions of our data are uncorrelated. We can then read off the variance in the different (orthogonal) directions. The key insight is that one can always write a set of n correlated gaussian random variables as linear combinations of n independent gaussian random variables (conversely, one can construct n independent gaussians out of linear combinations of n correlated gaussians). These linear combinations are the most natural axes with which to view our data.

This is all very well if our data originally had multivariate gaussian distribution. However we said at the beginning that they could have some complicated unknown joint probability density function. Indeed, much real world data does not follow a multivariate gaussian distribution. Nevertheless, PCA identifies the directions of greatest variance in a high dimensional dataset, by fitting a multivariate gaussian distribution to the empirically observed data.

1.1 Procedure

Let's say we measure n properties of a system t times. We can compile these measurements into a $n \times t$ matrix X where each column of X is a single measurement. For example, we could be measuring temperature, air pressure, humidity, wind speed etc. every day for a year. First we shift our data so that the mean of each row is 0, so our matrix corresponds to *deviations* in air pressure, humidity etc. from the mean (taken across time). Sometimes we will rescale our data so that all columns have unit variance — otherwise doing PCA with different units in different columns does not give results that are invariant to choice of units. Even if all columns have the same units then this rescaling may change the interpretation of results, and should be done with care (though is common practice).

Now we can calculate an empirical covariance matrix, which we will call $\hat{\Sigma}$. The elements of this matrix are defined by $\hat{\Sigma}_{ij} = \hat{\Sigma}_{ji} = \mathrm{E}[x_i x_j]$, where the expectation is taken over all t measurements. This means that

$$\hat{\Sigma} = \frac{1}{t} X X^\intercal. \tag{1}$$

Now we can diagonalize this matrix by writing it as $\hat{\Sigma} = U^{-1} \Lambda U$ where the columns of U are the eigenvectors of $\hat{\Sigma}$ and Λ is a diagonal matrix whose ith entry is the variance of the data along the direction given by the ith column of U. We can order the columns of U by the magnitudes of the eigenvalues, with the first column having the largest eigenvalue. The data will be most spread out when projected onto the column of U with the highest eigenvalue. If we want to keep just the first two principal components of the data and plot it in two dimensions, we can keep the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$, and make a scatter plot in two dimensions of $\mathbf{x}_i \cdot \mathbf{u}_2$ versus $\mathbf{x}_i \cdot \mathbf{u}_1$ for every i: these are the *projections* of our full dataset onto the first two principal components.

1.2 A Simple Example: Multivariate Gaussian

The simplest way to see PCA at work is if we simply draw the $\mathbf{x}_i$ from a multivariate gaussian distribution to begin with. In this case, if we have enough data, the empirically measured covariance matrix will converge to the true covariance matrix which generated the data. Upon calculating its eigenvectors, and ranking them in order of their eigenvalues from largest to smallest (all eigenvalues are nonnegative since covariance matrices are positive semidefinite) we obtain an orthonormal basis in which the variance of the data along the kth basis vector decreases with increasing k.

How do we know how well we've actually done in reducing the dimensionality of our dataset? One comparison we can make is against a *null model* that simply picks a set of orthogonal vectors at random from $\mathbb{R}^n$ and projects our data onto this basis. How does this totally naive method compare to our systematic PCA?

In Figure 1 we carry out this procedure for a data set where each point is 4 dimensional, and there are 5000 independent measurements of each point. To generate these data, we specified some "true" covariance matrix and then drew a large number of samples from the corresponding multivariate gaussian distribution. Therefore our data matrix is a 4×5000 matrix X, and we can readily calculate $\hat{\Sigma} = \frac{1}{5000} X X^\intercal$.

The normalized eigenvectors of $\hat{\Sigma}$ are the principal components of our dataset. Once we rank them by their corresponding eigenvalues from largest to smallest, we see, as expected, that the data has the largest variance along the direction of the 1st principal, the second largest along the 2nd, etc.

We can capture more than half of the variance of the dataset in the first principal component itself and therefore can reduce the dimensionality of our data from 4 to 1 while preserving most of the variance. In Figure 1 we show the results of keeping the first two principal components, which preserves even more of the variance. By comparison, if we conduct our dimensionality reduction by randomly picking two n dimensional vectors and orthogonalizing them via the Gram-Schmidt process, we see that we are unable to capture as much variance in our dimensionally-reduced data.

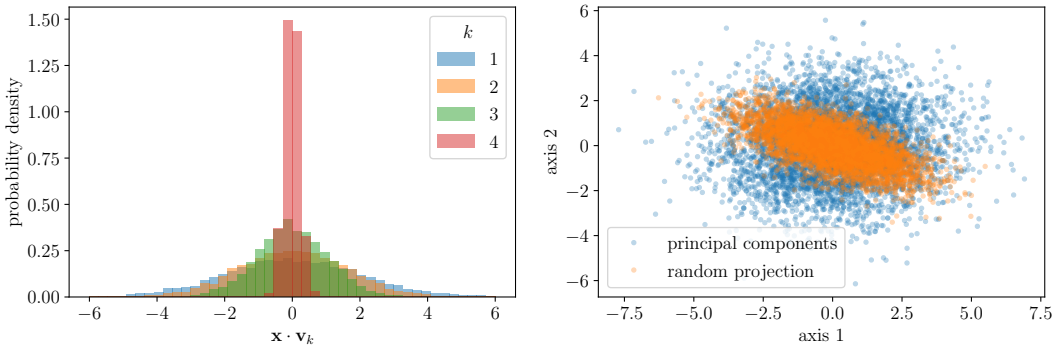

Figure 1: (A) Here we choose an arbitrary covariance matrix and generate data from a multivariate gaussian distribution. We can then calculate the empirical covariance matrix and diagonalize it to find the principal components of this distribution. We project the data onto each one of the principal components in turn by calculating $\mathbf{x}_i \cdot \mathbf{v}_k$ for principal components $\mathbf{v}_k$ and $k \in \{1, 2, 3, 4\}$, and see (as expected) that the variance of the projections decreases as the rank of the principal component increases (and the corresponding eigenvalue decreases). In this case it turns out the the first principal component explains slightly over 50% of the variance in the data, as calculated by the fraction $\lambda_1 / \sum_k \lambda_k$ for the eigenvalues of the empirical covariance matrix. (B) The results of our PCA are compared with a random projection. We see that although the first random axis captures a similar amount of variance to the first principal component, the second one captures substantially less.

One often sees PCA in conjunction with clustering algorithms. However PCA on its own is a dimensionality reduction technique — which does not inherently have to do with clustering data point according to their membership in various categories. However reducing dimensionality of data can make it easier to cluster and to visualize as we will see in the following example.

1.3 Another Example: Spike Sorting

We have seen that PCA (trivially) works on data that is multivariate gaussian distributed. What about data drawn from some other, more complicated distribution? Why should we use PCA as a dimensionality reduction tool in this case? And what does it do for us?

Say that our dataset $\{\mathbf{x}_i\}$ consists of a number of points in $\mathbb{R}^n$, each of which is distributed with equal probability according to 1 of k possible probability distributions. We would eventually like to say which data point was generated by which underlying distribution, but we are presented with an unlabeled dataset. A common case of this might be in the context of neural spike sorting, in which the goal is to cluster together neuron spikes of similar shape, based on the hypothesis that they are likely to have been produced by the same neuron. Since this task is not the main focus of these notes, we will not discuss in depth the biological motivations and relevant background for the

process of sorting spikes, but will rather view it simply as a data analysis task. For more information on the (large) field of spike sorting and electrophysiology, see the references at the end of these notes.

We can simulate some fake data corresponding to measurements of action potentials (spikes) of neurons, where each spike carries a label, corresponding to the neuron that produced it, and the spikes from different neurons are drawn from slightly different distributions corresponding to individual characteristics of different cells. We then can then reduce the dimensionality of our dataset by PCA and check whether the different classes of observation cluster in any self-evident way, when projected along the first few principal components.

In order to simulate firing neurons, we will use the Fitzhugh Nagumo model, a simplified version of the celebrated Hodkin-Huxley model for action potentials. The Fitzhugh Nagumo model is a slightly more complicated version of the van der Pol oscillator, and is a second order system described by two coupled ordinary differential equations. The model is

$$\dot{v} = v - \frac{v^3}{3} - w + RI + \sqrt{2D}\xi(t) \tag{2}$$

$$\tau\dot{w} = v + a - bw, \tag{3}$$

where I is the input stimulus, and $\xi(t)$ is δ-correlated white noise, whose inclusion ensures that a given set of parameters $\{\tau, R, a, b\}$ defines a *distribution* of spike profiles generated from these values. Here v is the observable variable (the voltage difference between the outside and inside of the neuron), and w is an auxiliary variable without a clear physical interpretation. Different neurons can be modeled by different parameters, which should produce differently shaped action potentials. For each set of parameters, the average spike shape corresponds to a "true" spike shape of that neuron (Figure 2).

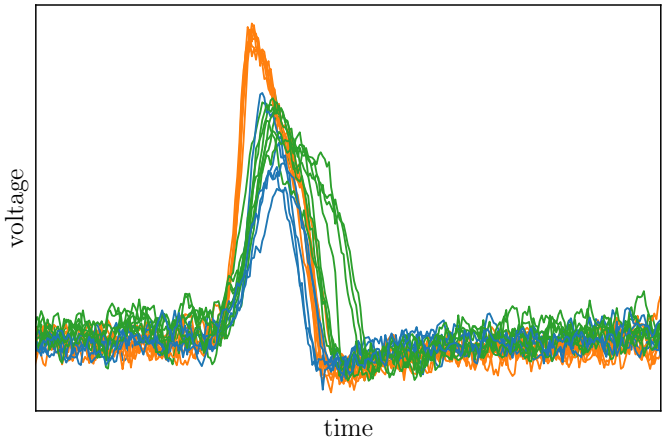

Figure 2: Numerical integration of the noisy Fitzhugh-Nagumo equations for 3 sets of parameters $\{\tau, R, a, b\}$, corresponding to 3 different neurons. Therefore each neuron is associated with a high dimensional probability distribution of the spike signal that it produces. Here the spikes from different neurons are shown in different colors.

We should note that although PCA will identify the directions of maximal variance of the data, these will not necessarily coincide with the directions along which different clusters are the most separated. Nonetheless, PCA is a very common tool used in spike sorting.

In this case each spike that we simulate consists of 300 numbers that represent the voltage over time. Our data matrix X has dimensions 300×5000, where we simulated 5000 spikes, each one generated by one of the three possible neurons. We can find the leading eigenvectors of the covariance matrix of our data, and take the dot product of each data point with the first two eigenvectors to obtain the principal components. Here we have normalized the rows of X to have unit variance, in addition to 0 mean.

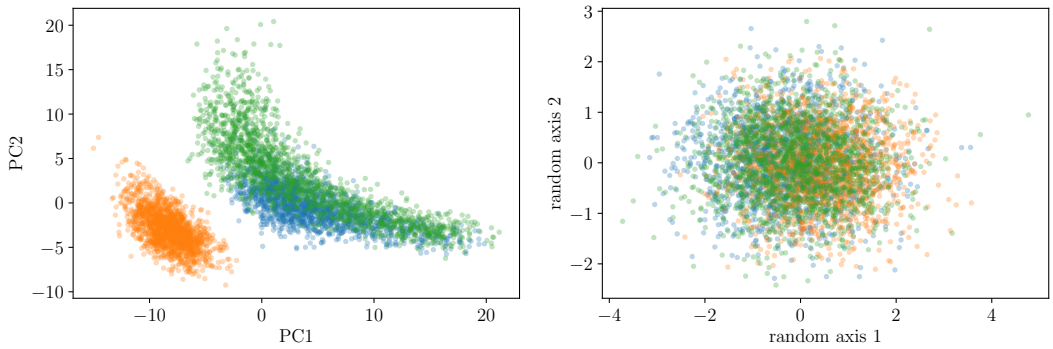

Figure 3: Here we see that the first two principal components already capture some of the data's structure. If we didn'tknow what color to make each data point (as would be the case for an empirical dataset), we would still see that there are empirically two clusters (at least). By contrast, the random dimensionality reduction shows no obvious clustering, and is therefore less informative as to the underlying structure of our data.

How did our PCA do? In Figure 3 we see that by projecting onto the first two principal components, we obtain a very clear separation into two clusters. The colors here correspond to the true labels, but this clustering would be obvious even if we did not have access to the true labels, as would be the case for a real dataset. As in the previous case, it is important to compare the results of the PCA to some kind of null model, wherein we pick two orthogonal vectors in $\mathbb{R}^{300}$ and use these as our principal components. In the second part of Figure 3 we see these results, and there are no obvious clusters at all. So it seems our PCA was effective at projecting the data into a subspace where the clustered structure is relatively evident.

1.4 k-means Clustering

We saw that PCA visually does a better job of separating our data into clusters than is accomplished by, for example, choosing a random plane that cuts through our data. But once we have conducted PCA and reduced the dimensionality of our dataset, how do we actually do the clustering? One widely used algorithm is called k-means clustering.

This algorithm is quite intuitive. The goal is to partition the data into k sets such that the sum of distances of each datapoint to the center of the set is minimized across all sets. The algorithm requires us to specify k, though in general we may not know the most appropriate k that would inform us about our dataset. There are other ways of determining this most appropriate k, but we will not discuss them here (see references for one example). In general finding this clustering is an NP-hard problem, but the naive k-means algorithm (which we use here) is one heuristic for trying to achieve this (though it is not guaranteed to yield an optimal answer). First we initialize the clusters with their centers located at k randomly chosen data points. Then we partition the data, assigning each point to the cluster whose center it is closest to. Having done this, we recalculate the center of each cluster. Then we iterate this algorithm until we have achieved a labelling of data points into clusters that is self consistent, i.e. the clustering does not change after we recalculate the centers of each cluster.

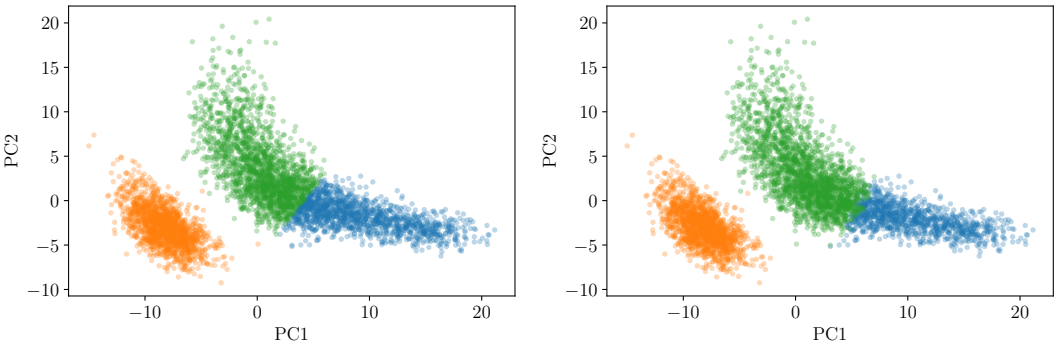

Figure 4: Results of k means cluster with $k = 3$. Each data point is colored according to the identity assigned to it by the k-means clustering algorithm.

In Figure 4A we see the results of a k-means cluster on our dataset with $k = 3$, projected onto its first 2 principal components. The algorithm captures the two obvious clusters well, and partitions the larger one into 2 as the true labels indicate it ought to. One could well ask why we bothered to go through the PCA in the first place — it's equally possible to run k-means clustering on our full high dimensional dataset. To assess what kind of a difference this would make, in Figure 4B we show the result of this procedure, where we kept all 300 principal components of our data and ran k-means clustering on the full data. There is not a large difference between these results and those from keeping only the first 2 principal components. Therefore PCA makes our clustering more interpretable and parsimonious (and potentially faster to compute for a really large dataset).

1.5 SVD

When you hear about PCA, it may be discussed in conjunction with another three letter acronym, SVD, which stands for singular value decomposition. However these really refer to the same process, and which one is used is largely a matter of preference. Doing SVD on the $n \times t$ matrix X is equivalent to diagonalizing its empirical covariance matrix, given by $\hat{\Sigma} = \frac{1}{t} XX^\intercal$. The difference is that the eigenvalues of $\hat{\Sigma}$ are the squares of the singular values of X. For computational purposes, it is possible to compute the SVD of X without calculating the entire matrix $\hat{\Sigma}$, and so in applications this may be preferable to computing the PCA as we have discussed here.

Formally, the SVD of a matrix X constitutes a factorization of the form $X = USW^\intercal$ where U is a $t \times t$ matrix consisting of orthonormal *left singular vectors* (eigenvectors of $X^\intercal X$), W is a $n \times n$ matrix consisting of orthonormal *right singular vectors* (eigenvectors of $XX^\intercal$) and S is a $t \times n$ matrix with its only nonzero elements on the diagonal: these elements, of which there are $\min(n, t)$, are called the singular values. Note that with this representation of X, we can straightforwardly calculate

$$XX^\intercal = USW^\intercal WS^\intercal U^\intercal = US^2U^\intercal, \tag{4}$$

where we used the fact that the columns of W are orthonormal in order to conclude that $W^\intercal W = I$, and where we denote $SS^\intercal$ by S^2. Note the resemblance that this form bears to an eigenvalue decomposition of the matrix $XX^\intercal$, where now S^2 is the diagonal matrix of eigenvalues of $XX^\intercal$.

2 A Null Model for Empirical Covariance Matrices

It we measure some multidimensional signal many times, we can always calculate an empirical covariance matrix which tells us the correlations between different components of the signal. How do we know whether these empirically measured covariances are actually significant? In other words, what is a null model for the empirical covariance matrix of a number of uncorrelated variables measured a finite number of times? This was a question answered by Marchenko and Pastur in 1967, that has relevance for our discussion.

Since we want to see what the correlations are that arise from t observations of n-dimensional random variable in the case that the n dimensions are all independent, it is natural to choose the random variable to be drawn from a multivariate gaussian distribution with 0 mean and covariance matrix Σ given by the $n \times n$ identity matrix. In the limit $t \to \infty$, the empirical covariance matrix, given by $\frac{1}{t} XX^\intercal$, would tend toward Σ. However for finite t there will be important effects. Already we know that if $t < n$ then A will not be full rank: in particular it will have $n - t$ zero-eigenvalues, since the dimension of its nullspace will be $n - t$. Therefore it is natural to numerically investigate the distribution of eigenvalues of A for $n < t$ and $n > t$.

We will do this in subsequent notes, and derive the distribution of the eigenvalues of $\hat{\Sigma}$ under the null model that X is a $n \times t$ matrix with uncorrelated normally distributed entries everywhere.

References

Article on spike sorting
Blog post on k-means and spike sorting, showing one way of determining k.