Principal Component Analysis

Ryan P. Adams

One of the challenges of the analysis of high-dimensional data is visualization. That is, we have a problem in which our data are, say, vectors of length 1000. We perhaps believe that there is interesting structure present, but we have no easy way to find it and study. The class of unsupervised learning techniques called *dimensionality reduction* provides one strategy for such visualization. The central idea is to find an alternative representation of the data that is, say, two dimensional. Our brains may have difficult time with 1000-dimensional data, but they are well-suited to reasoning about two or three dimensional spaces. The challenge is that there are many ways to turn 1000-dimensional data into two-dimensional data. Which one should we choose? Principal component analysis (PCA) provides one answer to that question. PCA is a classical technique for finding low-dimensional representations which are **linear projections** of the original data.

Concretely, let's imagine that we have N data, which are vectors $\boldsymbol{x}_n \in \mathbb{R}^D$. When we perform dimensionality reduction, we want to find some new set of representations $\{\boldsymbol{y}_n\}_{n=1}^N$ — one $\boldsymbol{y}_n \in \mathbb{R}^K$ for each of our original data $\boldsymbol{x}_n$ — with the property that $K < D$. Since PCA is a linear method, it looks for a good linear map $\boldsymbol{U}$ from $\mathbb{R}^D$ to $\mathbb{R}^K$. This is a projection since $K < D$. There are many such projections, however — as many as there are $K \times D$ matrices $\boldsymbol{U}$. Which one do we pick? One restriction is that PCA requires that the rows of $\boldsymbol{U}$ form an **orthonormal basis** for a K-dimensional subspace of $\mathbb{R}^D$. An orthonormal basis for $\mathbb{R}^D$ is a set of unit-length vectors $\{\boldsymbol{u}_d\}_{d=1}^D$, which are all orthogonal to each other. That is,

$$\boldsymbol{u}_d^\mathsf{T}\boldsymbol{u}_{d'} = \begin{cases} 1 & \text{if } d = d' \\ 0 & \text{if } d \neq d' \end{cases} . \tag{1}$$

If the rows of $\boldsymbol{U}$ are an orthonormal basis, then that means that $\boldsymbol{U}\boldsymbol{U}^\mathsf{T} = \mathbb{I}_K$, where $\mathbb{I}_K$ denotes the $K \times K$ identity matrix. The restriction to orthonormal matrices still doesn't quite get us a unique linear projection for the data, however. There are many such bases (effectively corresponding to many possible rotated K-dimensional subspaces) and we pick the one that best captures the structure of the data using ideas that come up frequently in unsupervised learning. In fact, there are two equivalent ways to find the PCA matrix, using two different criteria for what it means to find structure in data. Interestingly, it turns out that both of these views feel very much like the intuitions behind K-Means clustering! The main difference is that in K-Means we represented a datum by its closest cluster mean, whereas in PCA we represent each datum by a point in a lower-dimensional subspace.

Figures 1 and 2 should give you an idea as to how you might think about the results that PCA provides. In Figure 1, I took the voting records of 104 senators in the 113th US congress on 172 bills and used PCA to project them into two dimensions. I then plotted their names in the projected location and colored them according to party. If you're more into sports, Figure 2 shows the same kind of visualization, but using the per-game stats of a set of high-scoring NBA basketball players.

In both cases, you can see that the low-dimensional representation reveals structure in the space. People with similar behaviors tend to be located near each other.

Figures 3, 4, and 5 give a different view of the results from PCA. Here what I've done is plotted the principal component vectors as images – these are "directions" in image space. Figure 3 is the result of PCA run only on the "8" digits from MNIST. Figure 4 is the result of PCA on all of the MNIST digits. Figure 5 shows the same thing, but for the CIFAR-100 natural images.

1 The Reconstruction View of PCA

Recall that one of the ways we framed the clustering problem was in terms of reconstruction. This is an appealing view of unsupervised learning, and dimensionality reduction in particular, because it corresponds to learning a compression. That is, we are asking our learning procedure to find a compact representation of data such that it can reproduce the original version closely. We hope that if you can accurately reconstruct the data from the compressed representation, then somehow that compressed representation captures all of the most salient statistical properties of the data, i.e., the important stuff. K-Means clustering says that these important properties are well-captured by a nearby point $\boldsymbol{\mu}_k$, while principal component analysis says that these important properties are captured by a nearby point on a lower-dimensional subspace.

As described above, we have N data vectors $\boldsymbol{x}_n \in \mathbb{R}^D$. If we have an orthonormal basis $\{\boldsymbol{u}_d\}_{d=1}^D$, we can write each of these data as

$$\boldsymbol{x}_n = \bar{\boldsymbol{x}} + \sum_{d=1}^{D} \alpha_d^{(n)} \boldsymbol{u}_d \,, \tag{2}$$

where $\bar{\boldsymbol{x}}$ is the mean of the data:

$$\bar{\boldsymbol{x}} \equiv \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n \,, \tag{3}$$

and the $\alpha_d^{(n)}$ are the inner products of the (centered) data with each of the D basis vectors:

$$\alpha_d^{(n)} = (\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}} \boldsymbol{u}_d \,. \tag{4}$$

You can think of $\alpha_d^{(n)}$ as telling us how much of $\boldsymbol{x}_n$ is explained by the direction $\boldsymbol{u}_d$. Now, when we have all D of the $\alpha_d^{(n)}$ for a datum n, we can perfectly represent $\boldsymbol{x}_n$. However, what if we instead only had $K < D$ basis vectors? Then we probably wouldn't be able to represent $\boldsymbol{x}_n$ exactly and we'd wind up with an approximate version $\hat{\boldsymbol{x}}_n$:

$$\hat{\boldsymbol{x}}_n \equiv \bar{\boldsymbol{x}} + \sum_{d=1}^{K} \alpha_d^{(n)} \boldsymbol{u}_d \,. \tag{5}$$

How bad is this approximation? As in K-Means, we can pull our our trusty squared-error loss function and compare $\boldsymbol{x}_n$ and $\hat{\boldsymbol{x}}_n$:

$$J_n(\{\boldsymbol{u}_d\}_{d=1}^K) = (\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n)^2. \tag{6}$$

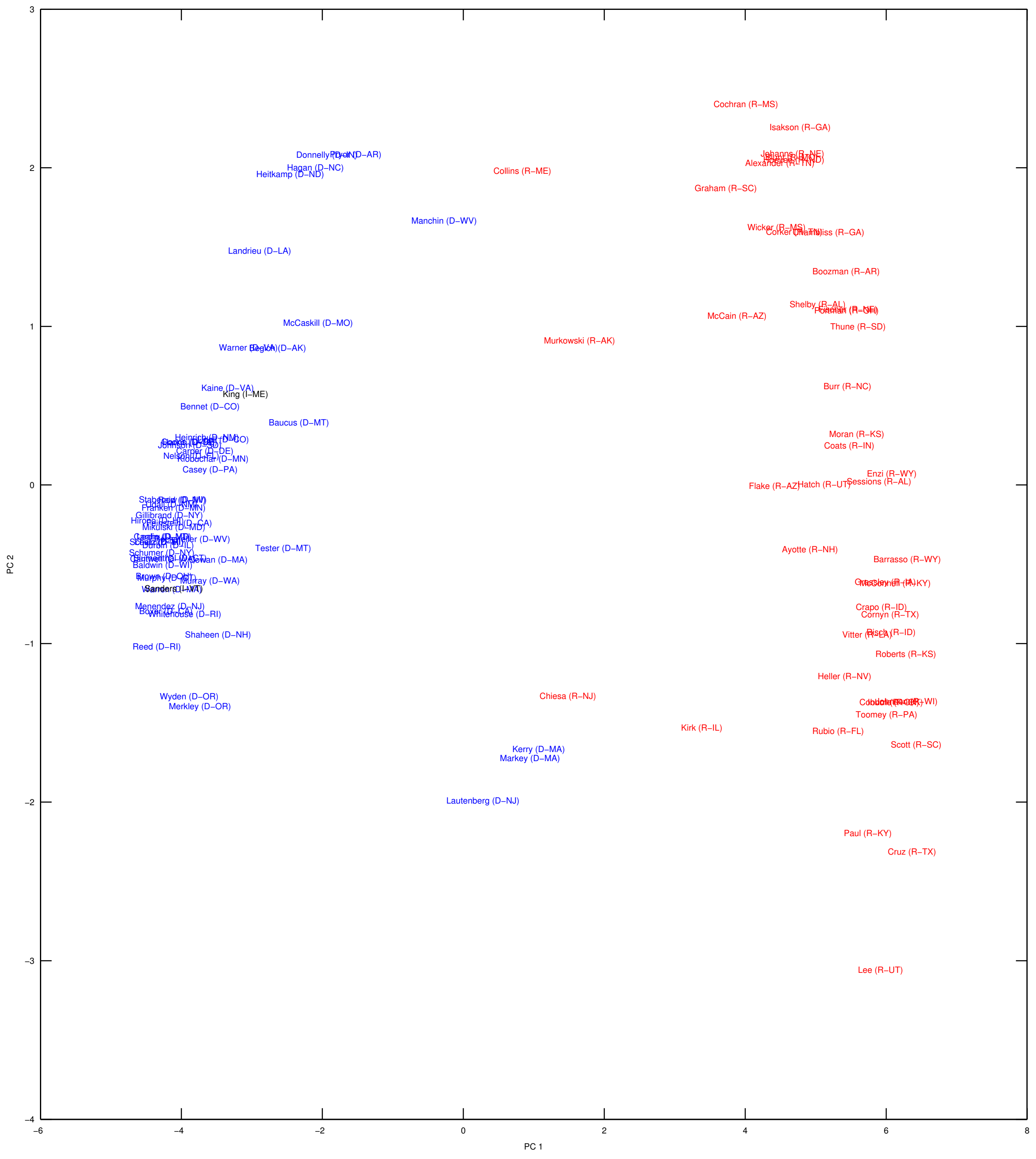


Figure 1: Two-dimensional representation of senators based on voting records. Parties are colored.

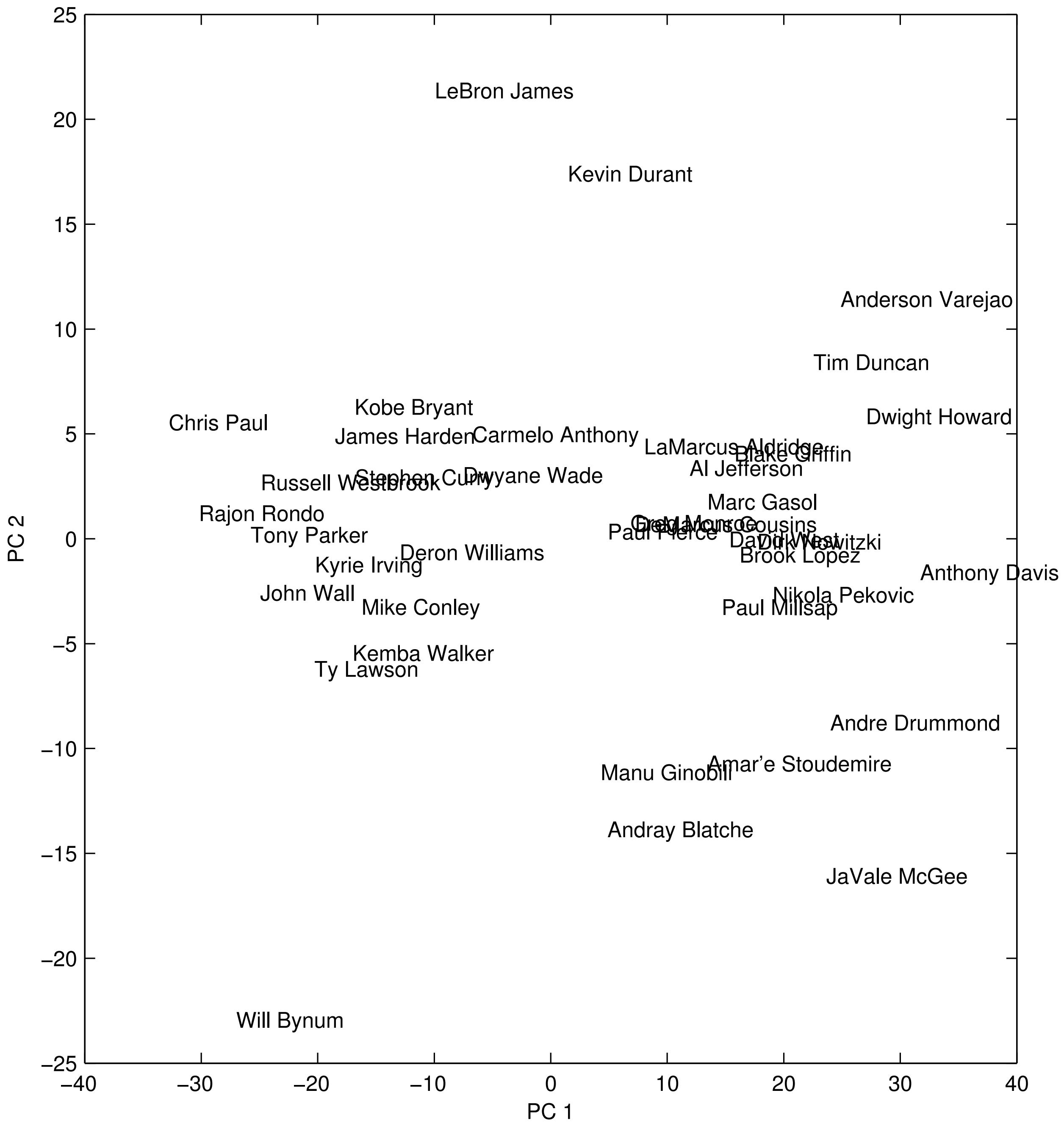


Figure 2: Two-dimensional representation of NBA basketball players.

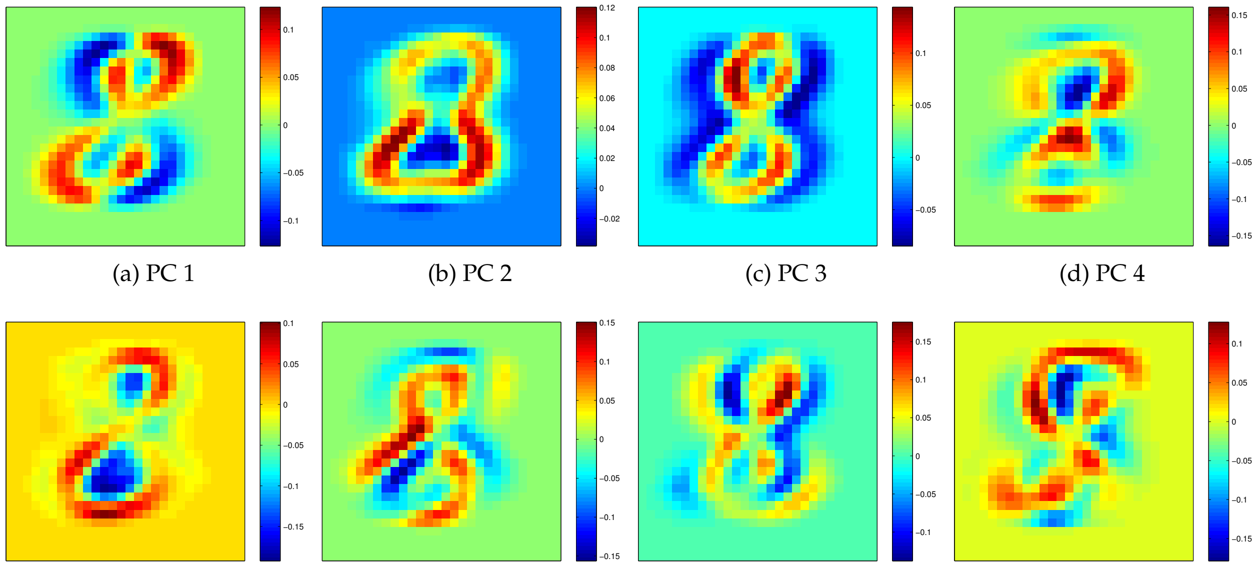


Figure 3: The first eight principal components of just the '8' digits from MNIST.

Now, we can plug our basis function expansions into both the original and its reconstruction:

$$\begin{aligned}
J_n(\{\boldsymbol{u}_d\}_{d=1}^K) &= \left(\left(\bar{\boldsymbol{x}} + \sum_{d=1}^{D} \alpha_d^{(n)} \boldsymbol{u}_d\right) - \left(\bar{\boldsymbol{x}} + \sum_{d=1}^{K} \alpha_d^{(n)} \boldsymbol{u}_d\right)\right)^2 && \text{Rewrite with mean and basis.} \\
&= \left(\left(\sum_{d=1}^{D} \alpha_d^{(n)} \boldsymbol{u}_d\right) - \left(\sum_{d=1}^{K} \alpha_d^{(n)} \boldsymbol{u}_d\right)\right)^2 && \text{The } \bar{\boldsymbol{x}}\text{'s cancel.} \\
&= \left(\sum_{d=K+1}^{D} \alpha_d^{(n)} \boldsymbol{u}_d\right)^2 && \text{The first } K \text{ bases are in both sums.}
\end{aligned}$$

Now we can take advantage of the orthonormality property (Equation 1) of the $\boldsymbol{u}_d$ vectors and just write this as:

$$J_n(\{\boldsymbol{u}_d\}_{d=1}^K) = \left(\sum_{d=K+1}^{D} \alpha_d^{(n)} \boldsymbol{u}_d\right)^2 = \sum_{d=K+1}^{D} \left(\alpha_d^{(n)}\right)^2. \tag{7}$$

At this point, you might be asking "Why are you writing the objective function on the left with the $\boldsymbol{u}_d$, when it doesn't appear on the right?". Let's fix that by plugging in Equation 4 and while we're at it, let's sum over all of the N data for the overall objective:

$$J(\{\boldsymbol{u}_d\}_{d=1}^K) = \sum_{n=1}^{N} \sum_{d=K+1}^{D} \left((\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\mathsf{T} \boldsymbol{u}_d\right)^2. \tag{8}$$

The intuition to have about this is that we're penalizing the distance between the subspace defined by $\{\boldsymbol{u}_d\}_{d=1}^K$ and the data.

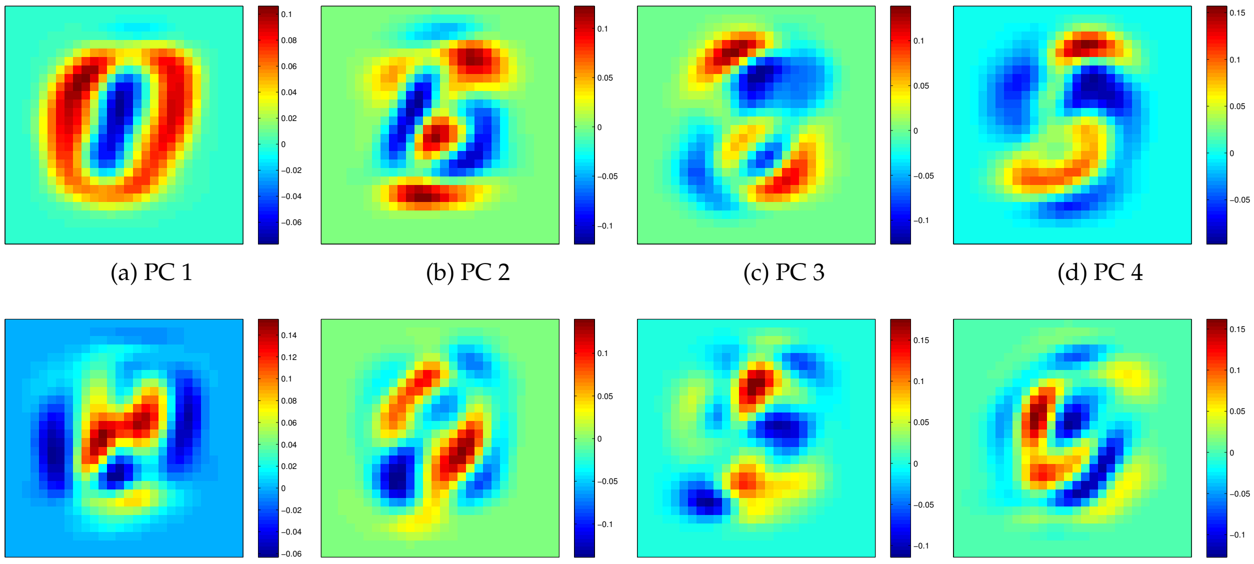

(a) PC 1 (b) PC 2 (c) PC 3 (d) PC 4
(e) PC 5 (f) PC 6 (g) PC 7 (h) PC 8

Figure 4: The first eight principal components of the MNIST training digits.

Let's write out the square in Equation 8 explicitly and transpose the first bit. It's just a scalar, so we can do this:

$$\begin{aligned}J(\{\boldsymbol{u}_d\}_{d=1}^K) &= \sum_{n=1}^N \sum_{d=K+1}^D \left((\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}} \boldsymbol{u}_d\right) \left((\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}} \boldsymbol{u}_d\right) \\ &= \sum_{n=1}^N \sum_{d=K+1}^D \boldsymbol{u}_d^{\mathsf{T}} (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}} \boldsymbol{u}_d \\ &= \sum_{n=1}^N \sum_{d=K+1}^D \boldsymbol{u}_d^{\mathsf{T}} \left((\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}}\right) \boldsymbol{u}_d \,.\end{aligned}$$

In this last step, I've put parentheses around the middle part, so that you can see it as a rank-one $D \times D$ matrix. It's a particularly interesting matrix, because it is the squared distance between each dimension of $\boldsymbol{x}_n$ and the mean vector $\bar{\boldsymbol{x}}$. We can swap the summations and push the sum over n inside the quadratic form. Let's multiply and divide by N, and push the $1/N$ inside the the quadratic form also:

$$J(\{\boldsymbol{u}_d\}_{d=1}^K) = N \sum_{d=K+1}^D \boldsymbol{u}_d^{\mathsf{T}} \left(\frac{1}{N} \sum_{n=1}^N (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}} \right) \boldsymbol{u}_d. \tag{9}$$

Suddenly now, we see that we have something really interesting here: that middle part is the sample covariance matrix![1] This is an important matrix, so let's give it its own symbol:

$$\boldsymbol{\Sigma} \equiv \frac{1}{N} \sum_{n=1}^N (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^{\mathsf{T}}. \tag{10}$$

[1]You could do this with $1/(N-1)$ and it would not make any practical difference.

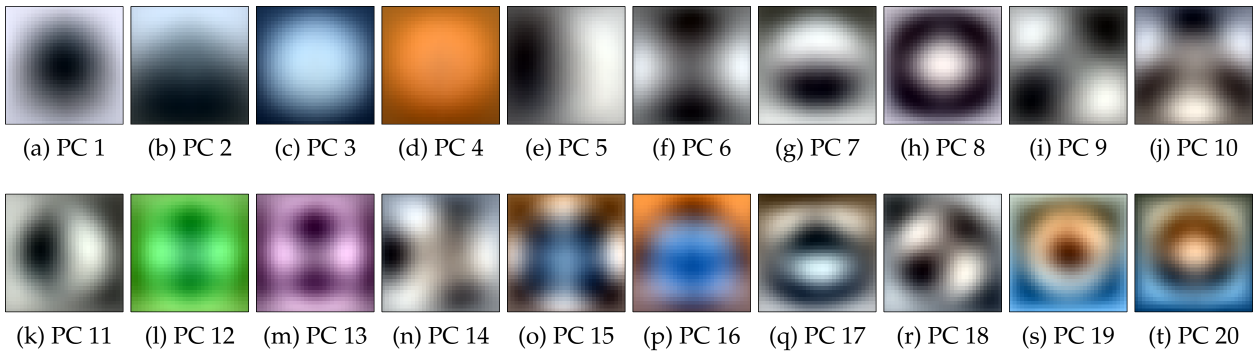


Figure 5: The first 20 principal components of the CIFAR-100 image data.

Recall that the objective here is to minimize J as a function of the K orthonormal vectors $\{\boldsymbol{u}_d\}_{d=1}^K$ that define our linear projection. We can write that down explicitly (the N doesn't matter because we're minimizing):

$$\min \left\{ \sum_{d=K+1}^{D} \boldsymbol{u}_d^\mathsf{T} \boldsymbol{\Sigma} \boldsymbol{u}_d \right\} \qquad \text{s.t. } \boldsymbol{u}_d^\mathsf{T} \boldsymbol{u}_d = 1 \,. \tag{11}$$

If we write this constrained objective using Lagrange multipliers, we get

$$N \sum_{d=K+1}^{D} \boldsymbol{u}_d^\mathsf{T} \boldsymbol{\Sigma} \boldsymbol{u}_d + \lambda_d (1 - \boldsymbol{u}_d^\mathsf{T} \boldsymbol{u}_d) \,. \tag{12}$$

If we differentiate this in terms of a $\boldsymbol{u}_d$, set it equal to zero and then solve, we see something very nice:

$$\boldsymbol{\Sigma} \boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d \,. \tag{13}$$

That is, the solutions will be eigenvectors of the sample covariance $\boldsymbol{\Sigma}$. Now that we know the orthonormality constraints will be satisfied, we can return to Equation 11:

$$\min \sum_{d=K+1}^{D} \boldsymbol{u}_d^\mathsf{T} \boldsymbol{\Sigma} \boldsymbol{u}_d = \min \sum_{d=K+1}^{D} \boldsymbol{u}_d^\mathsf{T} (\lambda_d \boldsymbol{u}_d) = \min \sum_{d=K+1}^{D} \lambda_d \,.$$

Here we substituted in Equation 13 and then exploited the orthonormality of $\boldsymbol{u}_d$ again. So, to minimize this we need to find the $D - K$ eigenvectors that have the smallest eigenvalues. That means that the K eigenvectors we pick for our projection should be the eigenvectors with the largest eigenvalues. Recall, by the way, that $\boldsymbol{\Sigma}$ is positive semi-definite, so all of the eigenvalues will be nonnegative.

In summary, to minimize the reconstruction error of a data approximation determined by a set of K orthonormal basis vectors, use the K eigenvectors of the sample covariance matrix with the largest eigenvalues.

2 The Variance-Preservation View of PCA

Reconstruction is a nice intuition, but it's not the only one. An alternative view of "salient statistical structure" in a set of data might simply be "variance". That is, when we find some lower-dimensional representation of the data, we want to preserve the bits that matter and interestingly differentiate things. Variability in the data is one such thing we might want to preserve. In the previous section, we computed the sample mean $\bar{\boldsymbol{x}}$ and the sample covariance matrix $\boldsymbol{\Sigma}$.

There are some identities for means and covariances that are very useful to know before we go on. Consider a random vector $\boldsymbol{z}$ that is drawn from a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{S}$. It turns out that if you hit this vector with a matrix $\boldsymbol{A}$ to get $\boldsymbol{A}\boldsymbol{z}$, that the result is *also Gaussian* and has mean $\boldsymbol{A}\boldsymbol{\mu}$ and covariance $\boldsymbol{A}\boldsymbol{S}\boldsymbol{A}^{\mathsf{T}}$.[2] This is useful because it helps us think about the properties of our projected data. Let's think about the first dimension of our projected data, $\boldsymbol{u}_1$, often called the first **principal component**. We want to choose this such that it maximizes the variance of the projected data. Due to this covariance transformation property, we know that the variance after projection will be $\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_1$. We want to maximize that, subject to the constraint that $\boldsymbol{u}_1$ be unit length:

$$\max \boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_1 \quad \text{s.t. } \boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_1 = 1\,. \tag{14}$$

We can invoke Lagrange multipliers again and look to maximize

$$\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_1 + \lambda_1(1 - \boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_1) \tag{15}$$

Taking the derivative in terms of $\boldsymbol{u}_1$ and setting it to zero, we see that at a stationary point

$$\boldsymbol{\Sigma}\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1 \tag{16}$$

so $\boldsymbol{u}_1$ must be an eigenvector. Furthermore,

$$\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_1 = \lambda_1\,, \tag{17}$$

meaning that this variance is maximized by choosing the eigenvector with the largest eigenvalue. For the second principal component, $\boldsymbol{u}_2$, we want to do the same thing, but make sure that our solution is also orthogonal to $\boldsymbol{u}_1$:

$$\max \boldsymbol{u}_2^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_2 \quad \text{s.t. } \boldsymbol{u}_2^{\mathsf{T}}\boldsymbol{u}_2 = 1 \text{ and } \boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_2 = 0\,. \tag{18}$$

We write down another Lagrangian:

$$\boldsymbol{u}_2^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_2 + \lambda_2(1 - \boldsymbol{u}_2^{\mathsf{T}}\boldsymbol{u}_2) + \gamma\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_2 \tag{19}$$

Differentiating in terms of $\boldsymbol{u}_2$, we get:

$$2\boldsymbol{\Sigma}\boldsymbol{u}_2 - 2\lambda_2\boldsymbol{u}_2 + \gamma\boldsymbol{u}_1 = 0\,. \tag{20}$$

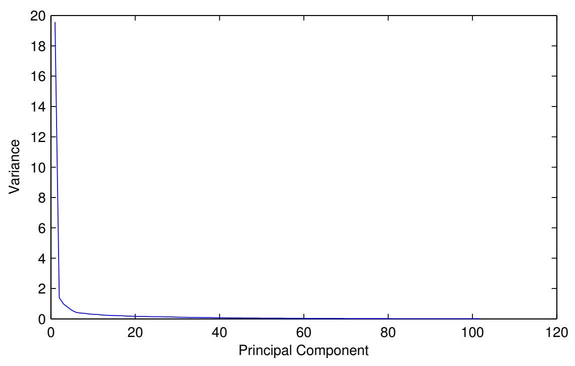
If we now left-multiply by $\boldsymbol{u}_1^{\mathsf{T}}$:

$$\boldsymbol{u}_1^{\mathsf{T}}\left(2\boldsymbol{\Sigma}\boldsymbol{u}_2 - 2\lambda_2\boldsymbol{u}_2 + \gamma\boldsymbol{u}_1\right) = 0 \tag{21}$$

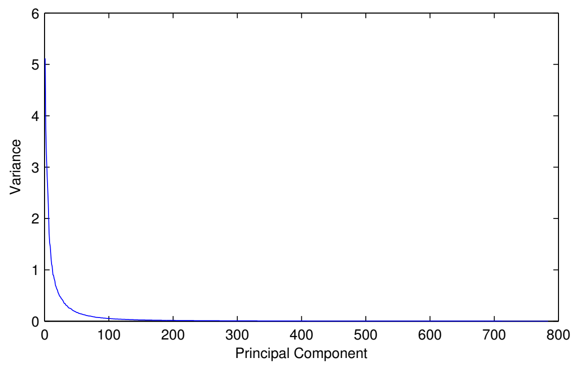
$$\cancel{2\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{u}_2} - \cancel{2\lambda_2\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_2} + \gamma\boldsymbol{u}_1^{\mathsf{T}}\boldsymbol{u}_1 = 0 \qquad \text{Due to orthonormality.} \tag{22}$$

$$\gamma = 0 \tag{23}$$

[2]For these and other useful Gaussian identities, see http://www.cs.nyu.edu/~roweis/notes/gaussid.pdf.



(a) PC vs. Variance of Senators



(b) PC vs. Variance of MNIST

Figure 6: These figures show the variance as a function of principal component. That is, the vertical axis is the eigenvalue associated with each eigenvector, going left to right. The drop-off of these plots gives us an idea as to how many "effective dimensions" there are in the data.

Thus the constrained solution must satisfy

$$\boldsymbol{\Sigma} \boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2 \tag{24}$$

and so $\boldsymbol{u}_2$ is also an eigenvector. We can do the same thing we did for $\boldsymbol{u}_1$ to convince ourselves that we want $\boldsymbol{u}_2$ to be the eigenvector with the *second* largest eigenvalue. We can iteratively perform this procedure for any $\boldsymbol{u}_k$ we want.

We can also think about this in a non-iterative way by thinking about how to maximize an aggregate measure of variance in the projection, rather than going one dimension at a time. When we have a covariance matrix $\boldsymbol{S}$, the **total variation** is the sum of the marginal variances of each dimension. It can easily be computed by looking at the trace of the matrix $\boldsymbol{S}$. When performing PCA one often discusses the "fraction of total variation" explained, which is the relative amount of total variation in the projection compared to the original data. The idea is that when one is explaining a large fraction of the variance with only a few principal components, then you have a good low-dimensional representation. Figure 6 provides one way to visualize this, by plotting variance (eigenvalue) as a function of principal component (eigenvector).

3 Singular Value Decomposition

Both of the approaches we have used so far to describe PCA have examined the spectrum of the covariance matrix. It's a little bit strange to do things, this way, however, because the covariance matrix is itself a product of matrices. To see this easily, let's imagine that we take all of our N data vectors $\boldsymbol{x}_n$, transpose them, and stack them to make a matrix $\boldsymbol{X}$ that is $N \times D$ where we'll be assuming $N > D$. Let's further imagine that we had zero-centered the data so that $\bar{\boldsymbol{x}} = 0$. Then the sample covariance matrix as we've been computing it here would be $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X}$. Singular value decomposition (SVD) is a way to arrive at the same answer as PCA, but without having to compute $\boldsymbol{\Sigma}$ first. As a result, SVD can be more numerically stable than PCA in practice.

Singular value decomposition computes the following factorization:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^{\mathsf{T}}, \quad \text{where} \tag{25}$$

$$\boldsymbol{U} \in \mathbb{R}^{N\times D} \quad \text{has orthonormal columns, i.e., } \boldsymbol{U}^{\mathsf{T}}\boldsymbol{U} = \mathbb{I}_D \tag{26}$$

$$\boldsymbol{\Gamma} \in \mathbb{R}^{D\times D} \quad \text{is diagonal with nonnegative entries} \tag{27}$$

$$\boldsymbol{V} \in \mathbb{R}^{D\times D} \quad \text{is orthonormal, i.e., } \boldsymbol{V}^{\mathsf{T}}\boldsymbol{V} = \mathbb{I}_D. \tag{28}$$

The diagonal entries of $\boldsymbol{\Gamma}$ are called the **singular values**. Note what happens when we write the covariance matrix for centered data:

$$\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X} \tag{29}$$

$$= \frac{1}{N}\left(\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^{\mathsf{T}}\right)^{\mathsf{T}}\left(\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^{\mathsf{T}}\right) \tag{30}$$

$$= \frac{1}{N}\boldsymbol{V}\boldsymbol{\Gamma}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^{\mathsf{T}} \tag{31}$$

$$= \frac{1}{N}\boldsymbol{V}\boldsymbol{\Gamma}^2\boldsymbol{V}^{\mathsf{T}}\,. \tag{32}$$

Thus the columns of $\boldsymbol{V}$ are the eigenvectors we are looking for with PCA and the square roots of the diagonal entries in $\boldsymbol{\Gamma}$ give us the eigenvectors, e.g., $\lambda_d = \Gamma_{d,d}/N$.

4 Advanced Topics

Factor Analysis: What if we relax the orthonormality constraint on the projection? Factor analysis is an alternative way to think about low-dimensional continuous structure in data. It has a long history in statistics. PRML discusses it a bit in Section 12.2.4. You can also check out the page on Metacademy.

Independent Component Analysis: When deriving principal component analysis, we made heavy use of squared errors and covariance matrices. Gaussian assumptions, however, mean that it will only be possible to recover latent structure up to a rotation. Independent component analysis (ICA) replaces Gaussian assumptions with heavier- or lighter-tailed distributions, making it possible to distinguish latent rotations. It is an important approach to difficult signal processing problems, such as *blind source separation*. PRML discusses this in Section 12.4.1 and Metacademy also has a page.

Multidimensional Scaling: When you want to cluster and you only have distances between objects, K-Medoids is an appealing alternative to K-Means. Is there an equivalent for PCA when you only have distances between your data? Multidimensional scaling (MDS) is one answer to this question. Rather than trying to preserve variance or minimize reconstruction error, it tries to find a low-dimensional embedding that preserves distances between data. PRML mentions MDS briefly in section 12.4.3, but there is more information on Metacademy and the Wikipedia page.

Nonlinear Dimensionality Reduction: There is an increasingly large body of work on nonlinear approaches to dimensionality reduction. Some important ideas in this area are Kernel PCA (See PRML 12.3), Isomap, Locally Linear Embedding, and t-SNE. Laurens van der Maaten maintains an excellent Matlab toolbox with implementations of many of these ideas.

Autoencoder Neural Networks: We often think of neural networks as a tool for supervised learning, e.g., predicting a label in visual object recognition task. However, we can ask a neural network to reproduce its input at its output instead, and turn it into an unsupervised learning tool called an *autoencoder*. This sounds like a boring thing to do, but if we limit its capacity in some way, then we force it to learn a compression of the data. In fact, it turns out that certain kinds of autoencoders learn exactly the same thing as PCA. A good resource for some of these ideas is DeepLearning.net. PRML also talks a bit about autoencoders in Section 12.4.2.