Contents

CHAPTER 1

Higher Dimensions

Up to this point, we have mainly discussed one dimensional data. This data is important, because it is common and easy to visualize and to deal with. Two dimensional data appeared when we talked about scatter plots and regression. However, much data has very high dimension. Representations of items like documents, images, or sound quite commonly involve thousands of dimensions.

High dimensions involve special problems of their own, which we describe in section **??**. There are relatively few reliable models we can use for high dimensional data, because there is often an unmanageable number of parameters to estimate. One quite reliable model is the multivariate normal distribution (or gaussian, as it is usually called). We describe these models in section **??**. Finally, from these models we derive a nice trick to reduce the dimension of a dataset in section **??**.

1.1 STRAIGHTFORWARD PROPERTIES OF HIGH DIMENSIONAL DATA

Generally, we will represent our data items with column vectors. The i'th data item is $\mathbf{x}_i$. If we need to refer to the j'th component of a vector $\mathbf{x}_i$, we will write $x_i^{(j)}$ (notice this isn't in bold, and the j is in parentheses because it isn't a power). In any data set, there are N such d-dimensional vectors.

There is a natural analogue to the mean in high dimensions. For one-dimensional data, we wrote

$$\mathsf{mean}\left(\{x_i\}\right) = \frac{\sum_i x_i}{N}.$$

This expression is meaningful for vectors, too, because we can add vectors and divide by scalars. We write

$$\mathsf{mean}\left(\{\mathbf{x}_i\}\right) = \frac{\sum_i \mathbf{x}_i}{N}$$

and call this the mean of the data. Notice that each component of $\mathsf{mean}\left(\{\mathbf{x}_i\}\right)$ is the mean of that component of the data. There is not an easy analogue of the median, however (how do you order high dimensional data?) and this is a nuisance. Notice that, just as for the one-dimensional mean, we have

$$\mathsf{mean}\left(\{\mathbf{x_i} - \mathsf{mean}\left(\{\mathbf{x}_i\}\right)\}\right) = 0$$

(i.e. if you subtract the mean from a data set, the resulting data set has zero mean).

If our data items are d dimensional vectors, we could compute

$$\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$$

for any pair of j, k. When j and k are different, this gives us the covariance of two components of the vectors. This behaves like the covariances we saw in Chapter 1.

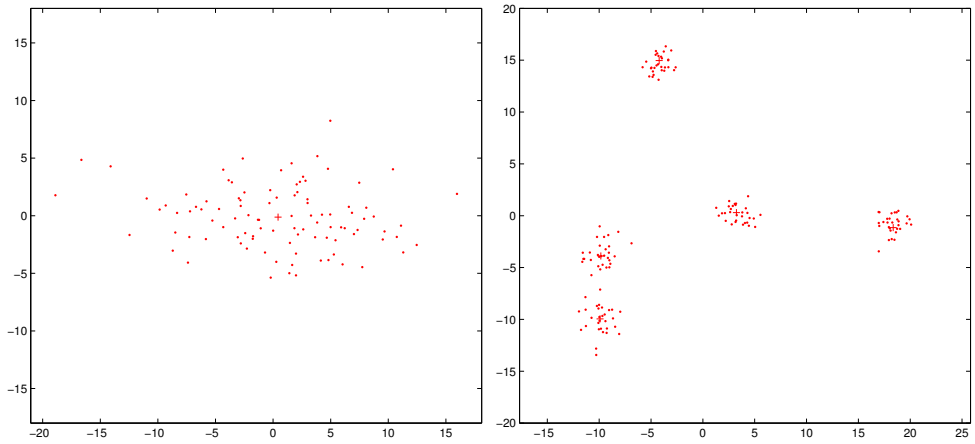

FIGURE 1.1: *On the* **left**, *a "blob" in two dimensions. This is a set of data points that lie somewhat clustered around a single center, given by the mean. I have plotted the mean of these data points with a '+'. On the* **right**, *a data set that is best thought of as a collection of five blobs. I have plotted the mean of each with a '+'. We could compute the mean and covariance of this data, but it would be less revealing than the mean and covariance of a single blob. In chapter 1.5, I will describe automatic methods to describe this dataset as a series of blobs.*

For example, if the j'th component tends to be positive when the k'th component is negative (and vice versa), $\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$ should be negative. Similarly, $\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right) = \mathsf{cov}\left(\left\{x^{(k)}\right\}, \left\{x^{(j)}\right\}\right)$. Finally, as we saw in Chapter 1, we must have that $\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(j)}\right\}\right) = \mathsf{std}\left(x^{(j)}\right)^2$, which we called **variance**.

There are $d(d-1)/2$ unique covariances, one for each j, k pair independent of order. Conveniently, we can compute all these covariances with one computation. We compute the matrix

$$\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right) = \frac{\sum_i (\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))^T}{N}$$

and the j, k'th entry of this matrix is $\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$, according to our original definition (you should check this sum).

When we plotted histograms, we saw that mean and variance were a very helpful description of data that had a unimodal histogram. If the histogram had more than one mode, one needed to be somewhat careful to interpret the mean and variance; in the pizza example, we plotted diameters for different manufacturers to try and see the data as a collection of unimodal histograms.

Generally, mean and covariance are a good description of data that lies in a "blob" (Figure 1.1). You might not believe that this is a technical term, but it's quite widely used. Mean and covariance are less useful as descriptions of data that forms multiple blobs (Figure 1.1). In chapter 1.5, we discuss methods to model data that forms multiple blobs, or other shapes that we will interpret as a set of blobs. But it's valuable to get some intuition for individual blobs, and we concentrate on

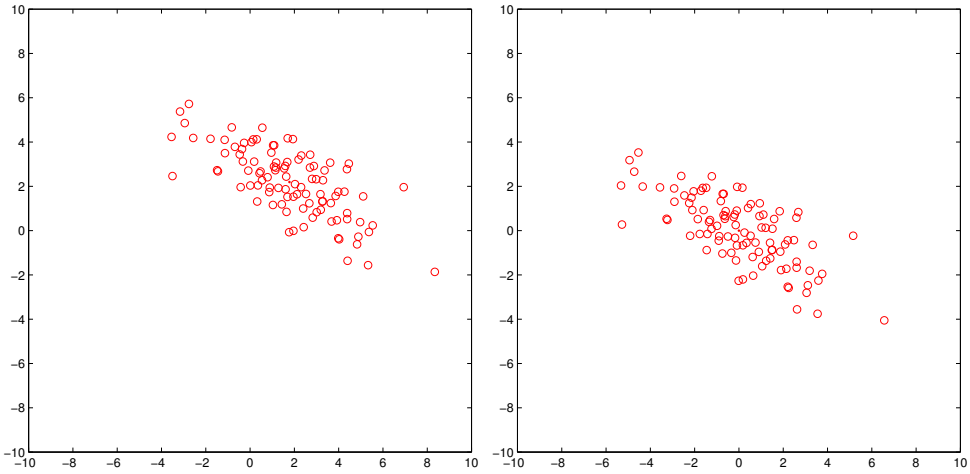

FIGURE 1.2: *On the* **left**, *a "blob" in two dimensions. This is a set of data points that lie somewhat clustered around a single center, given by the mean. I have plotted the mean of these data points with a '.' (it's easier to see when there is a lot of data). On the* **right**, *I subtracted the mean from each data point, which has the effect of translating the data — the whole blob is translated to lie around the origin.*

that here.

The trick to interpreting high dimensional data is to use the mean and covariance to understand the blob. Figure 1.2 shows a two-dimensional data set. Notice that there is obviously some correlation between the x and y coordinates (it's a diagonal blob), and that neither x nor y has zero mean. We can easily compute the mean and subtract it from the data points, and this translates the blob so that the origin is at the center (Figure 1.2).

If we were to rescale x and y to get normalized coordinates, we could compute the correlation between x and y. But there is another approach, which is more revealing. We can *rotate* the blob so that x and y are not correlated (Figure 1.3). The rotation can be read off the covariance matrix (next section), and it results in a new blob, in a new coordinate system, where there is no correlation between x and y. To describe this data, we need only specify the variance of x and the variance of y — the covariance is zero.

If we care to, we can now scale the data in this new coordinate system so that the variance of x and of y is one (Figure 1.4). This gives us a new type of normalized coordinate, where all covariances are zero, and all variances are one. There is a crucial point here: we can reduce any blob of data, in any dimension, to a standard blob of that dimension. All blobs are one, except for some stretching, some rotation, and some translation. This is why blobs are so well-liked.

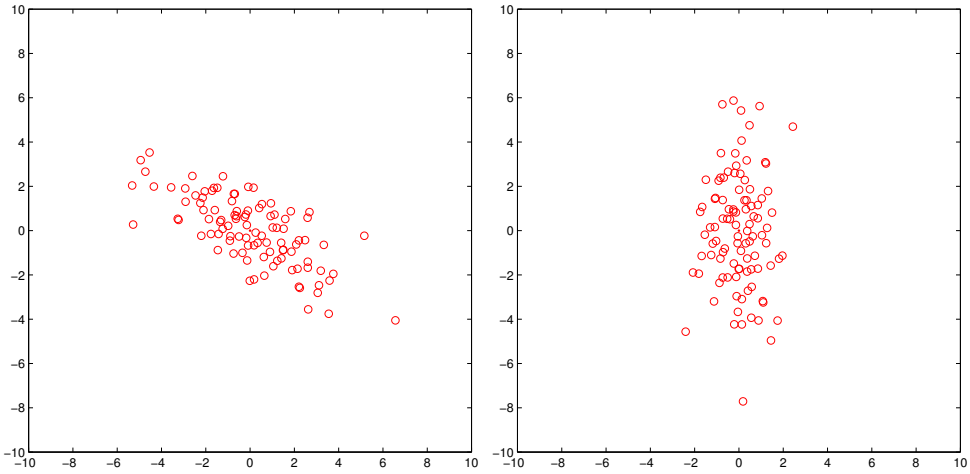

FIGURE 1.3: *On the* **left**, *the translated blob of figure 1.2. This blob lies somewhat diagonally, because the x and y components are correlated. On the* **right**, *that blob of data rotated so that there is no correlation. We can now describe the blob by the x and y variances — in the new coordinate system — alone.*

1.1.1 Linear Functions of High Dimensional Data

Consider a vector $\mathbf{v}$. We construct a new data set $u_i = \mathbf{v}^T\mathbf{x}_i$ — and these are not vectors, they are scalars. The data points in this new data set are linear functions of our original data points. Now we have

$$\begin{aligned}\mathsf{mean}\left(\{u_i\}\right) &= \mathsf{mean}\left(\left\{\mathbf{v}^T\mathbf{x}_i\right\}\right) \\ &= \mathbf{v}^T\mathsf{mean}\left(\{\mathbf{x}_i\}\right)\end{aligned}$$

and

$$\begin{aligned}\mathsf{cov}\left(\{u_i\},\{u_i\}\right) &= \frac{\sum_i (u_i - \mathsf{mean}\left(\{u_i\}\right))(u_i - \mathsf{mean}\left(\{u_i\}\right))}{N} \\ &= \frac{\sum_i (\mathbf{v}^T(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right)))(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))^T\mathbf{v}}{N} \\ &= \mathbf{v}^T\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)\mathbf{v}.\end{aligned}$$

This means that the covariance matrix must be positive semidefinite, because when we form $\mathbf{v}^T\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)\mathbf{v}$, the result is the variance of some data set (the u_i). Now assume that $\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)$ is indefinite — this means there is some non-zero $\mathbf{v}$ such that $\mathbf{v}^T\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)\mathbf{v} = 0$. But this means that

$$\sum_i \left[(\mathbf{v}^T(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right)))\right]\left[(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))^T\mathbf{v}\right] = 0$$

which means that

$$(\mathbf{v}^T(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))) = 0$$

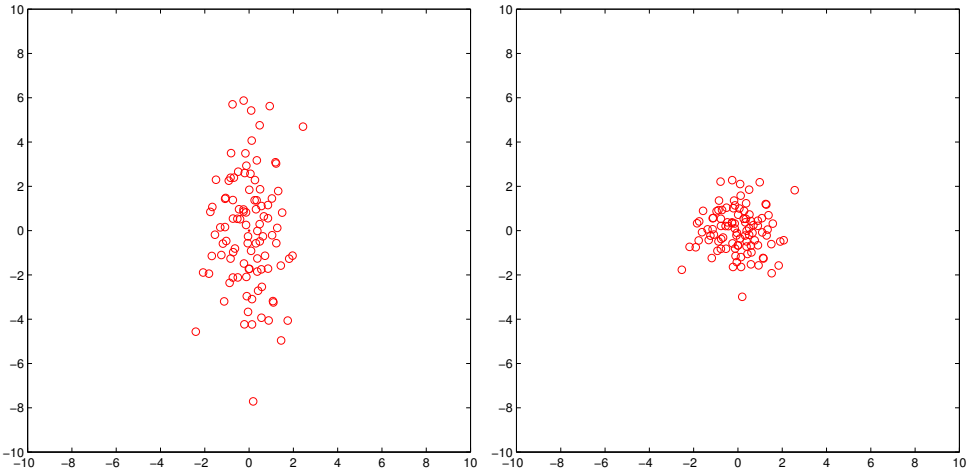

FIGURE 1.4: *On the* **left**, *the translated and rotated blob of figure 1.3. This blob is stretched — one direction has more variance than another. On the* **right**, *that blob of data scaled so that all variances are one. You can think of this as a standard blob. All blobs can be reduced to a standard blob, by relatively straightforward linear algebra (Section 1.1.1).*

for *every* i. In turn, for $\mathsf{Covmat}(\{\mathbf{x}_i\})$ to be indefinite, we must have that all the data points $\mathbf{x}_i$ lie on some hyperplane in the space (because $(\mathbf{v}^T(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}_i\}))) = 0$). Generally, this means that some entries in $\mathbf{x}_i$ are redundant. This case is easy to detect, and easy to correct if it occurs. We just have to project the data onto the hyperplane, to get a lower dimensional dataset. As a result, we generally assume that it doesn't happen (or, rather, has already been corrected). From now on, we will assume that $\mathsf{Covmat}(\{\mathbf{x}_i\})$ is positive definite (equivalently, you have projected off any bad dimensions already).

Useful facts: *Some Matrix Facts*

A matrix $\mathcal{M}$ is **symmetric** if $\mathcal{M} = \mathcal{M}^T$.
We write $\mathcal{I}$ for the identity matrix.
A matrix is **positive semidefinite** if, for any $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{x} > 0$ (i.e. this vector has at least one non-zero component), we have $\mathbf{x}^T\mathcal{M}\mathbf{x} \geq 0$.
A matrix is **positive definite** if, for any $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{x} > 0$, we have $\mathbf{x}^T\mathcal{M}\mathbf{x} > 0$.
A matrix $\mathcal{R}$ is **orthonormal** if $\mathcal{R}^T\mathcal{R} = \mathcal{I} = \mathcal{I}^T = \mathcal{R}\mathcal{R}^T$. You should think of orthonormal matrices as rotations, because they do not change lengths or angles. For $\mathbf{x}$ a vector, $\mathcal{R}$ an orthonormal matrix, and $\mathbf{u} = \mathcal{R}\mathbf{x}$, we have $\mathbf{u}^T\mathbf{u} = \mathbf{x}^T\mathcal{R}^T\mathcal{R}\mathbf{x} = \mathbf{x}^T\mathcal{I}\mathbf{x} = \mathbf{x}^T\mathbf{x}$. This means that $\mathcal{R}$ doesn't change lengths. For $\mathbf{y}$, $\mathbf{z}$ both unit vectors, we have that the cosine of the angle between them is $\mathbf{y}^T\mathbf{x}$; but, by the same argument as above, the inner product of $\mathcal{R}\mathbf{y}$ and $\mathcal{R}\mathbf{x}$ is the same as $\mathbf{y}^T\mathbf{x}$. This means that $\mathcal{R}$ doesn't change angles, either.

1.1.2 Affine Transformations of High Dimensional Data

Now assume we apply an affine transformation to our data $\mathbf{x}_i$, to obtain $\mathbf{u}_i = \mathcal{A}\mathbf{x}_i + \mathbf{b}$. Here $\mathcal{A}$ is any matrix (it doesn't have to be square, or symmetric, or anything else; it just has to have second dimension d). It is easy to compute the mean and covariance of $\mathbf{u}_i$. We have

$$\begin{aligned} \mathsf{mean}\left(\{\mathbf{u}_i\}\right) &= \mathsf{mean}\left(\{\mathcal{A}\mathbf{x}_i + \mathbf{b}\}\right) \\ &= \mathcal{A}\mathsf{mean}\left(\{\mathbf{x}_i\}\right) + \mathbf{b} \end{aligned}$$

and, which is much more interesting

$$\begin{aligned} \mathsf{Covmat}\left(\{\mathbf{u}_i\}\right) &= \mathsf{Covmat}\left(\{\mathcal{A}\mathbf{x}_i + \mathbf{b}\}\right) \\ &= \frac{\sum_i (\mathbf{u}_i - \mathsf{mean}\left(\{\mathbf{u}_i\}\right))(\mathbf{u}_i - \mathsf{mean}\left(\{\mathbf{u}_i\}\right))^T}{N} \\ &= \frac{\sum_i (\mathcal{A}\mathbf{x}_i + \mathbf{b} - \mathcal{A}\mathsf{mean}\left(\{\mathbf{x}_i\}\right) - \mathbf{b})(\mathcal{A}\mathbf{x}_i + \mathbf{b} - \mathcal{A}\mathsf{mean}\left(\{\mathbf{x}_i\}\right) - \mathbf{b})^T}{N} \\ &= \frac{\mathcal{A}\sum_i (\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right))^T \mathcal{A}^T}{N} \\ &= \mathcal{A}\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)\mathcal{A}^T. \end{aligned}$$

Useful facts: *Eigenvectors and Eigenvalues*

Assume $\mathcal{M}$ is a $d \times d$ matrix, $\mathbf{v}$ is a $d \times 1$ vector, and λ is a scalar. If we have

$$\mathcal{M}\mathbf{v} = \lambda\mathbf{v}$$

then $\mathbf{v}$ is referred to as an **eigenvector** of $\mathcal{M}$ and λ is the corresponding **eigenvalue**.
If $\mathcal{S}$ is a symmetric matrix, then it is possible to find a set of d eigenvectors that are normal to one another, and can be scaled to have unit length. They can be stacked into a matrix $\mathcal{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$. This matrix is orthonormal, meaning that $\mathcal{U}^T\mathcal{U} = \mathcal{I}$. You should think of $\mathcal{R}$ as a (high-dimensional) rotation of the coordinate system. In turn, this means that there is a diagonal matrix Λ such that

$$\mathcal{S}\mathcal{U} = \mathcal{U}\Lambda$$

equivalently, we have

$$\mathcal{U}^T\mathcal{S}\mathcal{U} = \Lambda.$$

This procedure is referred to as **diagonalizing** a matrix. In fact, we can do more. We can apply a transformation that transforms the matrix to an identity matrix, if $\mathcal{S}$ is positive definite. Then all diagonal elements of Λ are greater than 0. We can write $\Lambda^{1/2}$ for the diagonal matrix whose diagonal is the square roots of the diagonal elements of Λ. In Matlab, we could write `Lambda12=diag(((diag(Lambda)).^(1/2)))`. We have that $\Lambda^{1/2}\Lambda^{1/2} = \Lambda$. Furthermore, $(\Lambda^{1/2})^{-1}$ is the diagonal matrix whose diagonal elements are the reciprocal of the diagonal elements of $\Lambda^{1/2}$. In Matlab, we have `Lambda12inv=diag((1./((diag(Lambda)).^(1/2))))`. Then

$$(\Lambda^{1/2})^{-1}\mathcal{R}^T\mathcal{S}\mathcal{R}(\Lambda^{1/2})^{-1} = \mathcal{I}.$$

Now the covariance matrix is symmetric (check the facts if you have forgotten). This means that we can diagonalize it. In particular, let $\mathcal{U}$ be the matrix formed by stacking the eigenvectors of $\mathsf{Covmat}(\{\mathbf{x}_i\})$ into a matrix (i.e. $\mathcal{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$, where $\mathbf{v}_j$ are eigenvectors of the covariance matrix). Now choose $\mathcal{R} = \mathcal{U}^T$, and form $\mathbf{u}_i = \mathcal{R}\mathbf{x}_i$.

Then we have

$$\begin{aligned}
\mathsf{Covmat}(\{\mathbf{u}_i\}) &= \mathsf{Covmat}(\{\mathcal{R}\mathbf{x}_i\}) \\
&= \mathcal{R}\mathsf{Covmat}(\{\mathbf{x}_i\})\mathcal{R}^T \\
&= \mathcal{U}^T\mathsf{Covmat}(\{\mathbf{x}_i\})\mathcal{U} \\
&= \Lambda,
\end{aligned}$$

where Λ is a diagonal matrix of eigenvalues. This is an extremely important and useful idea, because the components of $\mathbf{u}$ are decorrelated from one another — every pair of distinct components has correlation zero. Another way to read this equation is that I can *rotate* the coordinate system of a set of high dimensional

data to put it in coordinates such that every component is decorrelated from every other component.

We could do more, because we assumed that $\mathsf{Covmat}(\{\mathbf{x}_i\})$ is positive definite. This means that we can compute the square roots of the diagonal matrix and transform the data into a coordinate system where the covariance matrix is the identity.

Useful facts: *Transforming high dimensional data*

Assume $\mathbf{x}_i$ has mean $\mathsf{mean}(\{\mathbf{x}_i\})$ and covariance matrix $\mathsf{Covmat}(\{\mathbf{x}_i\}) = \Sigma$ (we use Σ, for convenience; but it's also a convention) which is positive definite. Then we can compute $\mathcal{R}$, $\mathbf{b}$ so that

- $\mathbf{u}_i = \mathcal{R}(\mathbf{x}_i - \mathbf{b}) = \mathcal{R}\mathbf{x}_i - \mathcal{R}\mathbf{b}$
- $\mathsf{mean}(\{\mathbf{u}_i\}) = 0$,
- and $\mathsf{Covmat}(\{\mathbf{u}_i\}) = \Lambda$ (a diagonal matrix)

Notice the change of definition for $\mathbf{b}$ in the first item. I did this because we then have an easy expression for $\mathbf{b}$, which is $\mathbf{b} = \mathsf{mean}(\{\mathbf{x}_i\})$. As the text shows, $\mathcal{R} = \mathcal{U}^T$, where $\mathcal{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$, where $\mathbf{v}_j$ are eigenvectors of the covariance matrix Σ. Furthermore, we can compute $\mathcal{A}$, $\mathbf{b}$, so that

- $\mathbf{u}_i = \mathcal{A}(\mathbf{x}_i - \mathbf{b}) = \mathcal{A}\mathbf{x}_i - \mathcal{A}\mathbf{b}$
- $\mathsf{mean}(\{\mathbf{u}_i\}) = 0$,
- and $\mathsf{Covmat}(\{\mathbf{u}_i\}) = \mathcal{I}$ (i.e. the identity matrix).

Here $\mathcal{A} = (\Lambda)^{-1/2}\mathcal{R}$, where $\mathcal{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$.

1.2 PRINCIPAL COMPONENTS ANALYSIS

Imagine you have a high dimensional dataset that you wish to represent in a smaller set of dimensions. We will subtract the mean of the dataset, to simplify equations, so we assume that $\mathsf{mean}(\{\mathbf{x}_i\}) = 0$. There is no loss of generality here, it just means we can write simpler equations. For the moment, assume that you want to represent the data in one dimension. This means that you will represent this data by computing a projection of the data onto some direction, which we can write $\mathbf{v}$. We will write $u_i = \mathbf{v}^T\mathbf{x}_i$ for the projected data points. You should not scale the data when you project it, so we can insist that $\mathbf{v}^T\mathbf{v} = 1$. Which linear projection $\mathbf{v}$ should you choose?

For many applications, it is natural to choose the direction along which the data varies the most, that is, the direction with the highest projected variance. We know how to compute the variance in any particular direction. This is $\mathbf{v}^T\mathsf{Covmat}(\{\mathbf{x}_i\})\mathbf{v}$. So we could choose a direction to maximize this expression, subject to the constraint that $\mathbf{v}^T\mathbf{v} = 1$. The problem is to choose $\mathbf{v}$, without unnecessary mathematical fuss.

Choosing the direction in the right coordinate system is straightfor-

ward. The right system is the one where every pair of distinct components has covariance zero; we know how to rotate data into this coordinate system, using the eigenvectors of the covariance matrix. Write the data items in this coordinate system $\mathbf{y}_i = \mathcal{R}\mathbf{x}_i$. In this coordinate system, we have $\mathsf{Covmat}\left(\{\mathbf{y}_i\}\right) = \Lambda$, where Λ is a diagonal matrix. Now this means we have

$$\mathbf{v}^T \Lambda \mathbf{v} = \sum_i v_i^2 \lambda_i$$

(because Λ is diagonal). Remember that all λ_i must be positive, because $\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)$ is positive definite. The way to maximise this expression is to have $v_i = 1$ for i the index with the largest value of λ_i, and $v_i = 0$ for all others. We now have a (fairly obvious) result. In this coordinate system, if you want to the data with $k < d$ dimensions, while preserving the most variance, you choose the components corresponding to the k largest values of λ_i, and drop the rest. We could choose to represent on only one dimension — the dimension of the largest variance — but we could also choose to represent in k dimensions, where we would choose the k largest variance directions.

Solving in general is also straightforward. Our data does not usually start in this coordinate system. But we can easily compute a transformation into this coordinate system, which is the matrix $\mathcal{R}$, whose rows are eigenvectors of $\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)$. This works because the covariance of $\mathcal{R}\mathbf{x}_i$ is $\mathcal{R}\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)\mathcal{R}^T$, so if $\mathcal{R}^T$ has columns that are eigenvectors of $\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)$, the new covariance is diagonal.

Now imagine we choose the order of the rows in $\mathcal{R}$ to ensure that the diagonal of Λ is sorted in decreasing order. For this choice of $\mathcal{R}$, we have that the components of $\mathbf{y}_i = \mathcal{R}\mathbf{x}_i$ are ordered too. All pairs of distinct components have zero covariance. The first component has the largest variance, and the last component has the smallest variance. If we wanted to represent $\mathbf{y}_i$ in k dimensions, we would keep the first k components of $\mathbf{y}_i$.

One way to do this is to write $\mathcal{I}_k$ for the diagonal matrix whose diagonal is $(1, \ldots, 1(k \text{ times}), 0, \ldots, 0(d-k \text{ times}))$. The new representation is then $\hat{\mathbf{y}}_i = \mathcal{I}_k\mathbf{y}_i = \mathcal{I}_k\mathcal{R}\mathbf{x}_i$. But this is still in the $\mathbf{y}_i$ coordinate system. We know that $\mathbf{x}_i = \mathcal{R}^T\mathbf{y}_i$, so we could compute a new representation in the $\mathbf{x}_i$ coordinate system as

$$\hat{\mathbf{x}_i} = \mathcal{R}^T\mathcal{I}_k\mathcal{R}\mathbf{x}_i$$

Equivalently, we could write $\mathcal{R}_k$ for the $k \times d$ matrix consisting of the first k rows of $\mathcal{R}$. Then our new representation is

$$\hat{\mathbf{x}_i} = \mathcal{R}_k^T\mathcal{R}_k\mathbf{x}_i = \mathcal{M}_{svd}\mathbf{x}_i.$$

This expression is extremely rich. The $\hat{\mathbf{x}_i}$ lie on a k dimensional subspace of the original space, because $\mathcal{M}_{svd}$ has rank k. This k-dimensional space preserves the largest variance directions of the data. This is true in the $\mathbf{x}$ coordinate system because it was true in the $\mathbf{y}$ coordinate system, and all we did to get from one to the other was rotate. Another way to look at this is to notice that each of the $\hat{\mathbf{x}_i}$ is a weighted sum of k d-dimensional vectors (the columns of $\mathcal{R}_k^T$). This is why they lie on a k dimensional space. This k dimensional space is the best approximation

to the original data, in the sense of preserving the most variance. Each column of $\mathcal{R}_k^T$ is known as a **principal component**.

Procedure: *Principal Components Analysis*

Assume we have a general data set $\mathbf{x}_i$, consisting of N d-dimensional vectors. We compute $\hat{\mathbf{x}}_i = \mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}_i\})$, to obtain a zero-mean version of this dataset. Now write $\Sigma = \mathsf{Covmat}(\{\hat{x}_i\}) = \mathsf{Covmat}(\{\hat{\mathbf{x}}_i\})$ for the covariance matrix.
Form $\mathcal{U}$, Λ, such that $\Sigma\mathcal{U} = \mathcal{U}\Lambda$ (these are the eigenvectors and eigenvalues of Σ). Ensure that the entries of Λ are sorted; we will work with decreasing order. Choose k, the number of dimensions you wish to represent. Write $\mathcal{R} = \mathcal{U}^T$. Typically, we do this by plotting the eigenvalues and looking for a "knee" (Figure **??**). It is quite usual to do this by hand.
Constructing a low-dimensional representation: Form $\mathcal{R}_k$, a matrix consisting of the first k rows of $\mathcal{R}$. Now compute $\mathbf{n}_i = \mathcal{R}_k\hat{\mathbf{x}}_i$. This is a set of data vectors which are k dimensional, and where each component is independent of each other component (i.e. the covariances of distinct components are zero).
Smoothing the data: Form $\mathbf{s}_i = \mathsf{mean}(\{\mathbf{x}_i\}) + \mathcal{R}_k^T\mathbf{n}_i$. These are d dimensional vectors that lie in a k-dimensional subspace of d-dimensional space. The "missing dimensions" have the lowest variance, and are independent.

There are now two useful things that we can do with these principal components. First, we can use it to construct low-dimensional representations of the data. If we compute $\mathbf{n}_i = \mathcal{R}_k\mathbf{x}_i$, we have a k dimensional data set with special properties. Each component is independent of each other component (the covariances of distinct components are zero). And the components capture the most important variances in the original data set.

Alternatively, we could use it to construct smoothed versions of the original data. Smoothing is the process of suppressing small, irrelevant variations by exploiting multiple data items; there are many smoothing algorithms and procedures. We compute $\mathbf{s}_i = \mathcal{R}_k^T\mathbf{n}_i = \mathcal{R}_k^T(\mathcal{R}_k\mathbf{x}_i)$. This is a data set of the same dimension as the original data (d), but where the smallest variance components — which in many cases are irrelevant — have been removed. One way to think of this process is that we have chosen a low-dimensional basis that represents the main variance in the data rather well. It is quite usual to think of a data item as being given by a the mean plus a weighted sum of these basis elements. In this view, the first weight has larger variance than the second, and so on.

Example: Principal Components and Scatter Plots

Recall the height-weight data set of section **??** (from `http://www2.stetson.edu/~jrasp/data.htm`; look for bodyfat.xls). This is, in fact, a 16-dimensional dataset. The entries are (in this order): bodyfat; density; age; weight; height; adiposity; neck; chest; abdomen; hip; thigh; knee; ankle; biceps; forearm; wrist.

The mean of this dataset is in table 1.1. It is reasonable to think of this as the measurements of an "average person". We subtract the mean, and perform a principal component analysis, using the code of listing 1.5. Table 1.2 gives the

bodyfat	density	age	weight	height	adiposity	neck	chest
18.9385	1.0556	44.8849	178.9244	70.1488	25.4369	37.9921	100.8242

abdomen	hip	thigh	knee	ankle	biceps	forearm	wrist
92.5560	99.9048	59.4060	38.5905	23.1024	32.2734	28.6639	18.2298

TABLE 1.1: *The mean of the bodyfat.xls dataset. I don't know the units, though I'd guess that age is in years, weight is in pounds, height is in inches. I'm not sure about neck, chest, abdomen, etc.; they seem large to be in inches.*

0.0000	0.2403	0.6839	1.0655	1.4553	1.6715	1.9146	2.3892
3.3873	4.4466	6.7966	11.2635	12.2388	40.4327	177.1665	1139.1

TABLE 1.2: *The eigenvalues of the covariance matrix for the bodyfat data set, in order. The right way to think of these eigenvalues is to assume that we have translated the data set so that the mean is at the origin, then rotated it so that covariances between distinct components are zero. In this case, the eigenvalues represent the variances of each component of the rotated data. Notice how some components have large variance (e.g. the last one), but many have quite small variances. Some directions have three orders of magnitude more variance than others. In turn, this means we should be able to ignore some, or even many, directions in the data, and still represent it reasonably well.*

bodyfat	density	age	weight	height	adiposity	neck	chest
0.2124	-0.0005	0.9335	-0.1230	-0.0696	0.0339	0.0127	0.1108
0.7177	-0.0018	-0.3155	-0.3140	-0.2856	0.1145	-0.0545	0.1295
-0.5011	0.0012	0.0203	-0.0942	-0.7259	0.2258	0.0252	0.2356
0.1535	-0.0004	0.1391	0.0753	-0.1642	-0.0465	-0.0738	-0.7577
-0.3160	0.0007	-0.0506	-0.1130	0.2899	-0.0521	-0.1085	-0.1596

abdomen	hip	thigh	knee	ankle	biceps	forearm	wrist
0.1982	-0.0417	-0.0884	-0.0063	-0.0199	-0.0176	-0.0177	0.0099
0.3865	0.0753	0.1005	-0.0437	-0.0464	-0.0194	-0.0212	-0.0487
0.0912	0.2402	0.1913	-0.0187	-0.0189	0.0416	-0.0220	-0.0033
-0.1354	0.3807	0.3939	0.1353	0.0401	0.0040	-0.0662	-0.0085
0.6484	0.2825	-0.1304	-0.0281	-0.0701	-0.3845	-0.2977	-0.0386

TABLE 1.3: *The first five principal components of the bodyfat.xls data set, in order of decreasing variance. One way to interpret these is that you could model any element of the data set as the mean plus a weighted sum of these five principal components. By doing so, you would represent most of the variance in the data set. The remaining variance is less than 1% of the total variance; this means that this basis captures most of what is important in the original dataset. Notice that the first principal component can be interpreted as saying that the most important variation in the data is, approximately, in age; the second suggests that the next most important is in bodyfat, though bodyfat goes up as ages goes down.*

variances (equivalently, the eigenvalues of the covariance matrix). In the rotated coordinate system, each component is independent of each other component, but

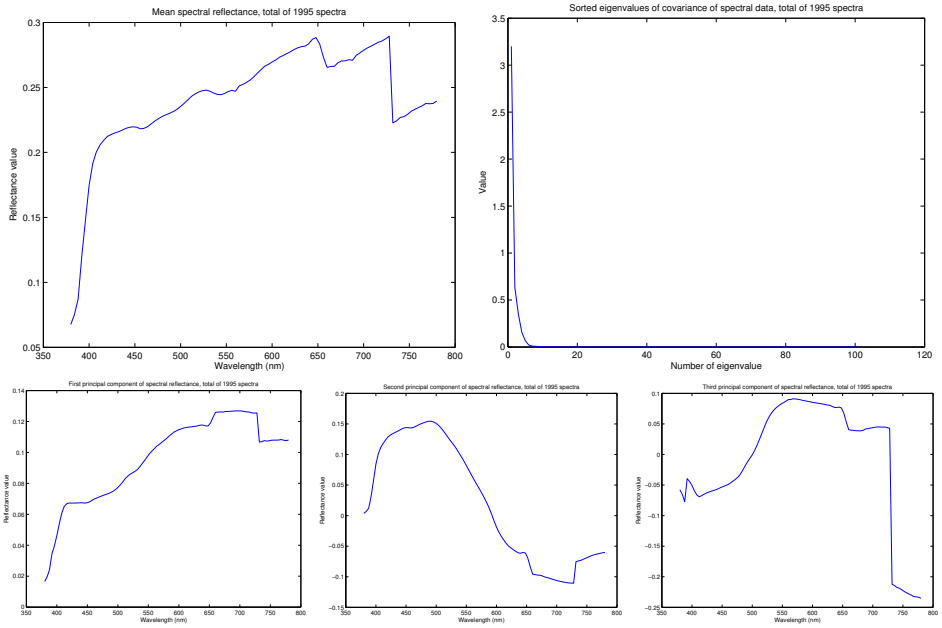

FIGURE 1.5: *On the* **top left**, *the mean spectral reflectance of a dataset of 1995 spectral reflectances, collected by Kobus Barnard (at* `http://www.cs.sfu.ca/~colour/data/`*). On the* **top right**, *eigenvalues of the covariance matrix of spectral reflectance data, from a dataset of 1995 spectral reflectances, collected by Kobus Barnard (at* `http://www.cs.sfu.ca/~colour/data/`*). Notice how the first few eigenvalues are large, but most are very small; this suggests that a good representation using few principal components is available. The* **bottom row** *shows the first three principal components. A linear combination of these, with appropriate weights, added to the mean of figure* **??**, *gives a good representation of the dataset.*

they have different variances. You can get some sense of the data by adding these variances; in this case, we get 1404. This means that, in the translated and rotated coordinate system, the average data point is about $37 = \sqrt{1404}$ units away from the center (the origin). Now translations and rotations do not change distances, so the average data point is about 37 units from the center in the original dataset, too.

The first five principal components are in table 1.3. These principal components correspond the the five largest eigenvalues. Now assume that we represent each data point as the mean plus appropriate weights times the first five principal components, as above. Then we will be ignoring the 11 smaller variance components, corresponding to approximately 17 units of variance; so the approximate data points will lie, on average, about $37 = \sqrt{1404 - 17}$ units away from the variance.

Example: Representing Spectral Reflectance with Principal Com-

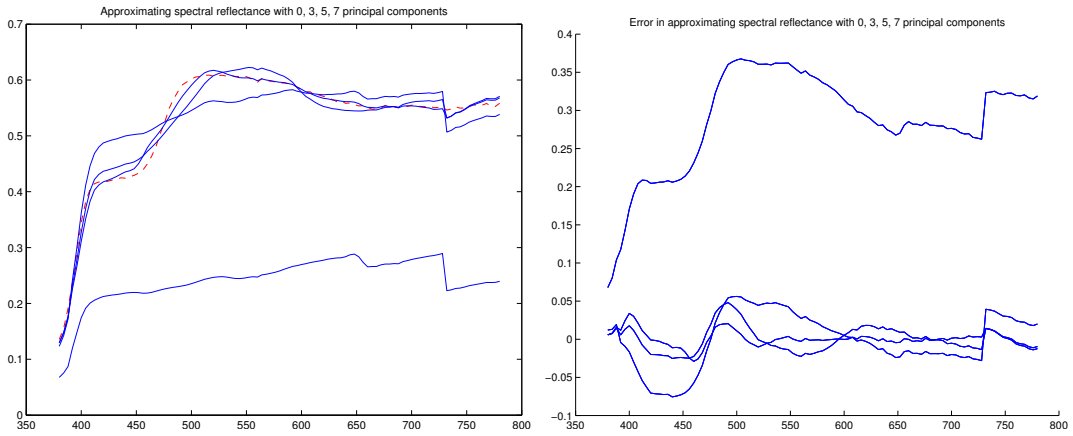

FIGURE 1.6: *On the* **left**, *a spectral reflectance curve (dashed) and approximations using the mean, the mean and 3 principal components, the mean and 5 principal components, and the mean and 7 principal components. Notice the mean is a relatively poor approximation, but as the number of principal components goes up, the error falls rather quickly. On the* **right** *is the error for these approximations. Figure plotted from a dataset of 1995 spectral reflectances, collected by Kobus Barnard (at* `http://www.cs.sfu.ca/~colour/data/`*).*

ponents

Diffuse surfaces reflect light uniformly in all directions. Examples of diffuse surfaces include matte paint, many styles of cloth, many rough materials (bark, cement, stone, etc.). One way to tell a diffuse surface is that it does not look brighter (or darker) when you look at it along different directions. Diffuse surfaces can be colored, because the surface reflects different fractions of the light falling on it at different wavelengths. This effect can be represented by measuring the spectral reflectance of a surface, which is the fraction of light the surface reflects as a function of wavelength. This is usually measured in the visual range of wavelengths (about 380nm to about 770 nm). Typical measurements are every few nm, depending on the measurement device. I obtained data for 1995 different surfaces from `http://www.cs.sfu.ca/~colour/data/` (great datasets here, from Kobus Barnard).

Each spectrum has 101 measurements, which are spaced 4nm apart. This represents surface properties to far greater precision than is really useful. Physical properties of surfaces suggest that the reflectance can't change too fast from wavelength to wavelength. It turns out that very few principal components are sufficient to describe almost any spectral reflectance function. Figure 1.5 shows the mean spectral reflectance of this dataset, and Figure 1.5 shows the eigenvalues of the covariance matrix.

This is tremendously useful in practice. One should think of a spectral reflectance as a function, usually written $\rho(\lambda)$. What the principal components analysis tells us is that we can represent this function rather accurately on a (really small) finite dimensional basis. This basis is shown in figure 1.5. This means that

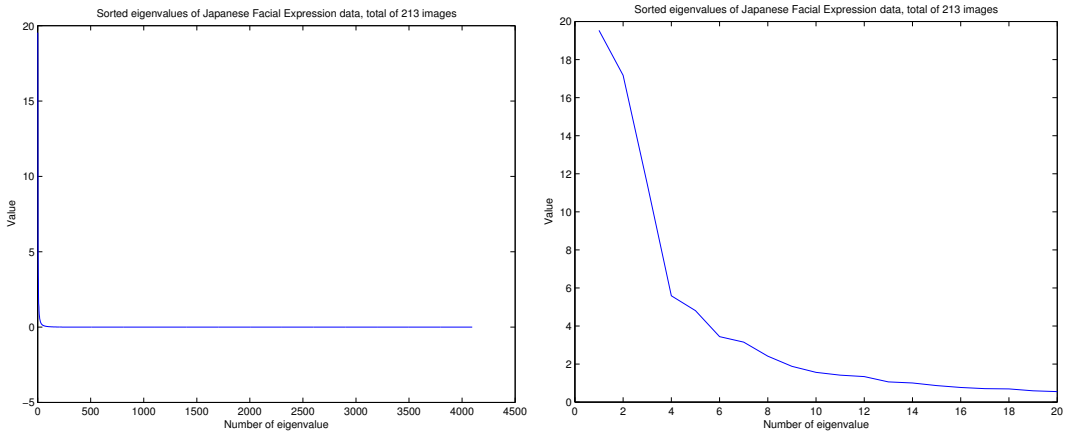

FIGURE 1.7: *On the* **left**,*the eigenvalues of the covariance of the Japanese facial expression dataset; there are 4096, so it's hard to see the curve (which is packed to the left). On the* **right**, *a zoomed version of the curve, showing how quickly the values of the eigenvalues get small.*

there is a mean function $r(\lambda)$ and k functions $\phi_m(\lambda)$ such that, for any $\rho(\lambda)$,

$$\rho(\lambda) = r(\lambda) + \sum_{i=1}^{k} c_i \phi_i(\lambda) + e(\lambda)$$

where $e(\lambda)$ is the error of the representation, which we know is small (because it consists of all the other principal components, which have tiny variance). In the case of spectral reflectances, using a value of k around 3-5 works fine for most applications (figure **??**).

Example: Representing Faces with Principal Components

An image is usually represented as an array of values. We will consider intensity images, so there is a single intensity value in each cell. You can turn the image into a vector by rearranging it, for example stacking the columns onto one another (use `reshape` in Matlab). This means you can take the principal components of a set of images. Doing so was something of a fashionable pastime in computer vision for a while, though there are some reasons that this is not a great representation of pictures. However, they give great intuition.

Figure **??** shows the mean of a set of face images encoding facial expressions of Japanese women (available at `http://www.kasrl.org/jaffe.html`; there are tons of face datasets at `http://www.face-rec.org/databases/`). I reduced the images to 64x64, which gives a 4096 dimensional vector. The eigenvalues of the covariance of this dataset are shown in figure 1.7; there are 4096 of them, so it's hard to see a trend, but the zoomed figure suggests that the first couple of hundred contain most of the variance.

Once we have constructed the principal components, they can be rearranged into images; these images are shown in figure 1.8. Principal components give quite

FIGURE 1.8: *The mean and first 16 principal components of the Japanese facial expression dataset.*

good approximations to real images (figure 1.9).

1.3 SIMILARITY AND SINGULAR VALUE DECOMPOSITIONS

In the previous section, we saw methods to represent high-dimensional datasets by analysing the covariance. Doing so allowed us to spot and exploit low dimensional structure in the data. There is another tool — the **singular value decomposition** — that can expose this structure as well.

Assume we have a matrix $\mathcal{D}$, which need not be symmetric or even square. We can decompose this matrix as

$$\mathcal{D} = \mathcal{U}\Sigma\mathcal{V}^T,$$

where $\mathcal{U}$ is orthonormal, $\mathcal{V}$ is orthonormal, and Σ is diagonal. This decomposition is the singular value decomposition, and it is a standard feature of numerical software. You should think of $\mathcal{U}$ and $\mathcal{V}$ as rotations in the output and input space of $\mathcal{V}$

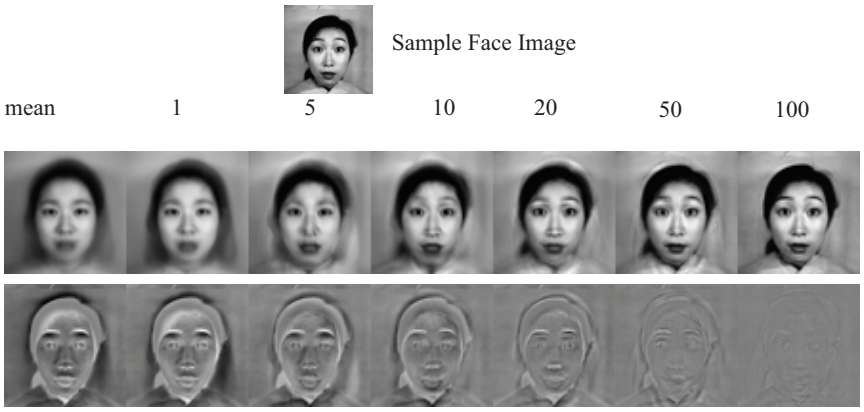

FIGURE 1.9: *Approximating a face image by the mean and some principal components; notice how good the approximation becomes with relatively few components.*

respectively. Now write $\mathbf{e}_1, \ldots, \mathbf{e}_n$ for a set of orthonormal vectors, such that $\mathcal{V} = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$. If $\mathbf{x}$ is a vector in the space spanned by this basis, we can represent $\mathbf{x}$ as

$$(\mathbf{e}_1^T\mathbf{x})\mathbf{e}_1 + (\mathbf{e}_2^T\mathbf{x})\mathbf{e}_2 + \ldots (\mathbf{e}_n^T\mathbf{x})\mathbf{e}_n.$$

In turn, the coefficients of this expansion

$$\begin{pmatrix} (\mathbf{e}_1^T\mathbf{x}) \\ (\mathbf{e}_2^T\mathbf{x}) \\ \ldots \\ (\mathbf{e}_n^T\mathbf{x}) \end{pmatrix} = \mathcal{V}^T\mathbf{x}.$$

Assume, for the moment, that $\mathcal{D}$ is square. When you form $\mathcal{D}\mathbf{x} = \mathcal{U}\Sigma\mathcal{V}^T\mathbf{x}$, you rotate $\mathbf{x}$ into a special coordinate system (by multiplying by $\mathcal{V}^T$), scale each component of the result, then rotate again (by multiplying by $\mathcal{U}$).

Now assume that $\mathcal{D}$ is not square, and is (say) $m \times n$. Then $\mathcal{V}$ is $n \times n$ and $\mathcal{U}$ is $m \times m$. Σ must be $m \times n$. Assume $m < n$. Write Σ_m for an $m \times m$ diagonal matrix, and 0 for an $m \times n - m$ matrix of zeros. Then Σ must look like

$$[\Sigma_n, 0]\,.$$

This means you can think about the singular value decomposition as follows: Rotate $\mathbf{x}$ into the special coordinate system, then scale each component of the result, dropping $n - m$ of them completely, then rotate the result of that step into a new coordinate system.

Similarly, if $m > n$, Σ must look like

$$\begin{bmatrix} \Sigma_n \\ 0^T \end{bmatrix}.$$

and this means that you can think about the singular value decomposition as follows: Rotate $\mathbf{x}$ into the special coordinate system, then scale the first n components

of the result, replacing the others with zero, then rotate the result of that step into a new coordinate system.

The diagonal elements of Σ, of which there are $s = \min(m, n)$, are called **singular values**. The following operation is of great importance. These are $\text{diag}(\Sigma) = (\sigma_1, \sigma_2, \ldots, \sigma_s)$, and we assume that they are sorted in descending order. We compute Σ_k so that $\text{diag}(\Sigma_k) = (\sigma_1, \sigma_2, \ldots, \sigma_k, 0, 0, \ldots, 0)$. Equivalently, we keep the singular values with the k largest magnitudes, and set all others to zero (we must have $k \leq s$). Now we construct $\mathcal{D}_k = \mathcal{U}\Sigma_k\mathcal{V}^T$. This matrix is an approximation to $\mathcal{D}$, where we zero the directions in the special coordinate system that are scaled by the smallest amounts.

Useful facts: *Singular Value Decomposition*

The singular value decomposition computes a decomposition of a matrix $\mathcal{D}$ into the form $\mathcal{D} = \mathcal{U}\Sigma\mathcal{V}^T$, where $\mathcal{U}$ and $\mathcal{V}$ are orthonormal, and Σ is diagonal.
It is usual to construct Σ so that the diagonal values, known as singular values, are sorted in descending order. There are $s = \min(m, n)$ singular values. The singular values are $\text{diag}(\Sigma) = (\sigma_1, \ldots, \sigma_s)$.
Write Σ_k for the matrix obtained by replacing the smallest $s - k$ singular values with zero, and $\mathcal{D}_k = \mathcal{U}\Sigma_k\mathcal{V}^T$. Then we have that $\mathcal{D}_k$ has rank k. It is the rank k matrix that is closest to $\mathcal{D}$ in the sense of squared distance.
If $\mathcal{M}$ is symmetric, then the singular value decomposition of $\mathcal{M}$ has the form $\mathcal{U}\Sigma\mathcal{U}^T$.

With some work (which we won't do), one can use this information to establish the most important property of the singular value decomposition: $\mathcal{D}_k$ is the best rank k approximation to $\mathcal{D}$. In particular, $\mathcal{D}_k$ is the rank k matrix with the smallest value of $\|\mathcal{D}_k - \mathcal{D}\|$ (sum of squared differences of the elements).

This is why SVD's are important. Assume that you know, perhaps from application logic, that you are dealing with a $\mathcal{D}$ that should have low rank. However, the $\mathcal{D}$ that you have consists of measurements, and so may have a higher rank than it should, as a result of measurement error. You can then easily find the low rank matrix that is closest to the measurements that you have, which you can reasonably expect is a correction of the data.

There are a variety of situations where you can expect to encounter low rank matrices. They are particularly important in information retrieval. Typical text information retrieval systems expect a set of query words. They use these to query some form of index, producing a list of putative matches. From this list they chose documents with a large enough similarity measure between document and query. These are ranked by a measure of significance, and returned.

1.3.1 Word Counts, Documents and Matching

Much of text information retrieval is shaped by the fact that a few words are common, but most words are rare. The most common words—typically including "the," "and," "but," "it"—are sometimes called **stop words** and are ignored because almost every document contains many of them. Other words tend to be rare, which means that their frequencies can be quite distinctive. Quite often, it is enough to know whether the word is there or not. For example, documents containing the

words "stereo," "fundamental," "trifocal," and "match" are likely to be about 3D reconstruction; documents containing "chrysoprase," "incarnadine," "cinnabarine," and "importunate" are lists of 11 letter words ending in "e" (many such lists exist, for crossword puzzle users; you can check this using Google).

Indexing Documents

It is straightforward to build a table representing the documents in which each word occurs, because very few words occur in many documents, so the table is sparse. Write N_w for the number of words and N_d for the number of documents. We could represent the table as an array of lists. There is one list for each word, and the list entries are the documents that contain that word. This object is referred to as an **inverted index**, and can be used to find all documents that contain a logical combination of some set of words. For example, to find all documents that contain any one of a set of words, we would: take each word in the query, look up all documents containing that word in the inverted index, and take the union of the resulting sets of documents. Similarly, we could find documents containing all of the words by taking an intersection, and so on. Such logical queries usually are not sufficient, because the result set might be either very large or too small, and because we have no notion of which elements of the result set are more important. We need a more refined notion of similarity between documents, and between a document and a query.

Similarity from Word Counts

One measure of similarity for two documents is to compare word frequencies. Assume we have a fixed set of terms that we will work with. We represent each document by a vector $\mathbf{c}$, with one entry for each term. These entries are zero when the term is absent, and contain some measure of the word frequency when the word is present. This measure might be as simple as a one if the word appears at least once in the document, or might be a count of the number of words. Write $\mathbf{c}_1$, $\mathbf{c}_2$ for two such vectors; the **cosine similarity** between the documents they represent is

$$\frac{\mathbf{c}_1 \cdot \mathbf{c}_2}{\|\mathbf{c}_1\| \|\mathbf{c}_2\|}.$$

Two documents that both use an uncommon word are most likely more similar than two documents that both use a common word. We can account for this effect by weighting word counts. The most usual way to do this is called **tf-idf weighting** (for "term frequency-inverse document frequency"). Terms that should have most weight appear often in the particular document we are looking at, but seldom in all documents. Write N_d for the total number of documents and N_t for the number of documents that contain the particular term we are interested in. Then, the inverse document frequency can be estimated as $N_d/(1+N_t)$ (where we add one to avoid dividing by zero). Write $n_t(j)$ for the number of times the term appears in document j and $n_w(j)$ for the total number of words that appear in that document. Then, the tf-idf weight for term t in document j is

$$\left(\frac{n_t(j)}{n_w(j)}\right) / \log\left(\frac{N_d}{(1+N_t)}\right).$$

We divide by the log of the inverse document frequency because we do not want very uncommon words to have excessive weight. Inserting this tf-idf weight into the count vectors above will get a cosine similarity that weights uncommon words that are shared more highly than common words that are shared.

1.3.2 Smoothing Word Counts and Latent Semantic Analysis

Our measurement of similarity will not work well on most real document collections, even if we weight by tf-idf. This is because words tend to be rare, so that most pairs of documents share only quite common words, and so most pairs of documents will have quite small cosine similarity. The real difficulty here is that zero word counts can be underestimates. For example, a document that uses the words "elephant," "tusk," and "pachyderm" should have some affinity for "trunk." If that word does not appear in the document, it is an accident of counting. This means that to measure similarity, we would do well to smooth the word counts.

We can do so by looking at how all terms are distributed across all documents. An alternative representation of the information in an inverted index is as an N_w by N_d table $\mathcal{D}$, where each cell contains an entry if the relevant word is not in the relevant document and a zero otherwise. Entries could be one if the word occurs, or a count of the number of times the word occurs, or the tf-idf weight for the term in the document. In any case, this table is extremely sparse, so it can be stored and manipulated efficiently. A column of the table is a representation of the words in a document, and the cosine similarity between columns is our original measure of similarity between documents.

Zeros in $\mathcal{D}$ might be the result of counting accidents, as above. We would like a version of this table that smooths word counts. There are likely to be many documents for any particular topic in the collection, so the smoothed version of the table should have many columns that are similar. This means it will be significantly rank-deficient. We compute a singular value decomposition of $\mathcal{D}$ as $\mathcal{D} = \mathcal{U}\Sigma\mathcal{V}^T$. Write $\mathcal{U}_k$ for the matrix consisting of the first k columns of $\mathcal{U}$, $\mathcal{V}_k$ for the matrix consisting of the first k columns of $\mathcal{V}$, Σ_k for Σ with all but the k largest singular values set to be zero, and write $\hat{\mathcal{D}} = \mathcal{U}_k\Sigma_k\mathcal{V}_k^T$.

Now consider the ith column of $\mathcal{D}$, which we write as $\mathbf{d}_i$. The corresponding column $\hat{\mathbf{d}}_i$ of $\hat{\mathcal{D}}$ lies in the span of $\mathcal{U}_k$. The word counts are smoothed by forcing them to lie in this span. For example, assume that there are many documents discussing elephants, and only one uses the word "pachyderm." The count vectors for each of these documents could be represented by a single column, but error will be minimized if there is a small count for "pachyderm" in each. Because of this smoothing effect, cosine distances between documents represented by columns of $\hat{\mathcal{D}}$ are a much more reliable guide to similarity.

To compute cosine similarity between an old document and a new document with count vector $\mathbf{q}$, we project the new document's count vector onto the columns of $\mathcal{U}_k$ to obtain $\hat{\mathbf{q}} = \mathcal{U}_k\mathcal{U}_k^T\mathbf{q}$. We can then take the inner product of $\hat{\mathbf{q}}$ and $\hat{\mathbf{d}}_i$. A complete table of inner products (cosine distances) between documents is given by

$$\hat{\mathcal{D}}^T\hat{\mathcal{D}} = (\mathcal{V}_k\Sigma_k)(\Sigma_k\mathcal{V}^T) = (\Sigma_k\mathcal{V}_k^T)^T(\Sigma_k\mathcal{V}^T),$$

so that we can think of the columns of $\Sigma_k\mathcal{V}^T$ as points in a k-dimensional "concept

space" that represents the inner products exactly. One could, for example, cluster documents in this space rather than the original count space, and expect a better clustering. Computing the SVD of $\mathcal{D}$ is known as **latent semantic analysis**; using the concept space for indexing is known as **latent semantic indexing**.

$\hat{\mathcal{D}}$ is useful in other ways, too. There is a rough tendency of words that have similar meaning to appear near similar words in similar documents, an idea known as **distributional semantics**. This means that cosine similarity between *rows* of $\hat{\mathcal{D}}$ is an estimate of the similarity of meaning of two terms, because it counts the extent to which they co-occur. Furthermore, $\hat{\mathcal{D}}$ can be used as an inverted index. If we use it in this way, we are not guaranteed that every document recovered contains all the words we used in the query; instead, it might contain very similar words. This is usually a good thing. The columns of $\mathcal{U}$ are sometimes called **topics** and can be thought of as model word frequency vectors; the coordinates of a column in the semantic space show the weights with which topics should be mixed to obtain the document.

1.3.3 Recommender Systems

Online marketing presents sellers with an important problem: it can be hard for a buyer to tell what is available, because there is so much stuff. Most of this stuff is only interesting to some other buyer. Sellers would like to have systems that can suggest new purchases (or rentals, or links to click on, and so on) to buyers. These systems need to be accurate, and to handle a very large number of preferences.

Assume there are N_c customers and N_i items. We can build an N_i by N_c table $\mathcal{D}$, containing preference scores of buyers for items. We might get these scores by asking for reviews, or by surveying the buyers. Assume that buyers do rate items, and tell the truth when they do so (neither is a minor assumption). If this table were complete, we would expect it to have relatively low column rank, because one can reasonably expect that there are many buyers who have many similar preferences. You can replace "buyer" with "renter", etc., here and the argument is still sound. The problem is that the table isn't complete — we are missing ratings for a particular buyer for some items, because these are the ratings we want to predict. In fact, we are missing ratings for many buyers for many items, because no-one buys everything.

So there is a matrix $\mathcal{M}$, known to have low column rank and known to be similar to $\mathcal{D}$ *in the entries of* $\mathcal{D}$ *that we know*. This matrix $\mathcal{M}$ represents all the ratings we expect. If we could predict this matrix, then we have an estimate of everyone's rating for each product. Although $\mathcal{M}$ has $N_i \times N_c$ entries, they are not independent because $\mathcal{M}$ has rank k. In fact, knowing only $N_i \times k + k \times N_c$ numbers is enough to yield $\mathcal{M}$. This is because there must be an $N_i \times k$ matrix $\mathcal{A}$ and a $k \times N_c$ matrix $\mathcal{B}$ so that

$$\mathcal{M} = \mathcal{A}\mathcal{B}.$$

We cannot expect to compute an SVD of $\mathcal{D}$, because we don't know every entry of $\mathcal{D}$. But we could compute a low-rank approximation. Write δ_{ij} for a variable that is 1 if we know the i, j'th value of $\mathcal{D}$ and 0 if we don't. Recall we write d_{ij} for the i, j'th entry in $\mathcal{D}$. Assume that $\mathcal{M}$ has rank k, which we know from our understanding of the problem, or by cross-validation (i.e. try several different k's

and see how good the predictions are). We now seek an $N_i \times k$ matrix $\mathcal{A}$ and a $k \times N_c$ matrix $\mathcal{B}$ so that

$$\mathcal{M} = \mathcal{A}\mathcal{B} \text{ is like } \mathcal{D} \text{ in the entries we know}$$

and we can get this by choosing $\mathcal{A}$, $\mathcal{B}$ to minimize

$$C(\mathcal{A}, \mathcal{B}) = \sum_{ij} \left[\delta_{ij} \left(\sum_l a_{il} b_{lj} - d_{ij} \right)^2 \right].$$

It turns out that this is quite easy to do. We construct initial estimates of $\mathcal{A}$ and $\mathcal{B}$, which we call $\mathcal{A}^{(0)}$ and $\mathcal{B}^{(0)}$. Now we compute the $i+1$'th estimate of $\mathcal{A}$ by minimizing

$$C(\mathcal{A}, \mathcal{B}^{(i)})$$

as a function of $\mathcal{A}$ alone. We then compute the $i+1$'th estimate of $\mathcal{B}$ by minimizing

$$C(\mathcal{A}^{(i+1)}, \mathcal{B})$$

as a function of $\mathcal{B}$ alone. Each step involves solving a linear system, and so is relatively straightforward.

Once we have $\mathcal{A}$, $\mathcal{B}$, we form $\mathcal{M} = \mathcal{A}\mathcal{B}$. This is now a low-rank matrix that is close to $\mathcal{D}$ in the entries that we know. Each term in $\mathcal{M}$ should be a good prediction of a particular buyer's rating for the relevant item.

Of course, this discussion has been highly simplified. There are usually so many items and so many buyers that one would want to modify the method rather a lot. It should strike you that we should not need to recompute $\mathcal{A}$, $\mathcal{B}$ every time a buyer rates an item; that we haven't explained how to deal with new buyers; and that there may be other measures of similarity between buyers that cue us to which columns of the matrix should be similar.

1.3.4 Multidimensional Scaling

The general problem of finding embeddings for points in some dimension so that the distances are similar to a given table of distances is known as **multidimensional scaling**. Assume we have n points we wish to embed in an r-dimensional space. Write $\mathcal{D}^2$ for a table of *squared* distances between points, with d_{ij}^2 the squared distance between point i and point j. Notice that if $\mathcal{D}^2$ is a table of distances, it will be symmetric. Write the embedding for point i as $\mathbf{x}_i$. Because translation does not change the distances between points, we can choose the origin, and we will place it at the mean of the points, so that $\frac{1}{n}\sum_i \mathbf{x}_i = \mathbf{0}$. Write $\mathbf{1}$ for the n-dimensional vector containing all ones, and $\mathcal{I}$ for the identity matrix. By noticing that $d_{ij}^2 = \| \mathbf{x}_i - \mathbf{x}_j \|^2 = \mathbf{x}_i \cdot \mathbf{x}_i - 2\mathbf{x}_i \cdot \mathbf{x}_j + \mathbf{x}_j \cdot \mathbf{x}_j$, we can show that

$$\mathcal{M} = -\frac{1}{2} \left[\mathcal{I} - \frac{1}{n} \mathbf{1}\mathbf{1}^T \right] \mathcal{D}^2 \left[\mathcal{I} - \frac{1}{n} \mathbf{1}\mathbf{1}^T \right]$$

has i, jth entry $\mathbf{x}_i \cdot \mathbf{x}_j$. This means that, to estimate the embedding, we must obtain a matrix $\mathcal{X}$ whose columns are the embedded points, so that $\mathcal{M}$ is "close"

to $\mathcal{X}^T\mathcal{X}$. A variety of notions of "closeness" might be appropriate; the easiest to use is least squares. In this case, we can apply a singular value decomposition to $\mathcal{M}$ to get $\mathcal{M} = \mathcal{U}\Sigma\mathcal{U}^T$. We form $\mathcal{G} = \Sigma^{1/2}\mathcal{U}^T$, and the first r rows of $\mathcal{G}$ are the $\mathcal{X}$ we require.

1.4 THE CURSE OF DIMENSION

High dimensional models display uninituitive behavior (or, rather, it can take years to make your intuition see the true behavior of high-dimensional models as natural). In these models, most data lies in places you don't expect. We will do several simple calculations with an easy high-dimensional distribution to build some intuition.

Assume our data lies within a cube, with edge length two, centered on the origin. This means that each component of $\mathbf{x}_i$ lies in the range $[-1, 1]$. One simple model for such data is to assume that each dimension has uniform probability density in this range. In turn, this means that $P(x) = \frac{1}{2^d}$. The mean of this model is at the origin, which we write as $\mathbf{0}$.

The first surprising fact about high dimensional data is that most of the data can lie quite far away from the mean. For example, we can divide our dataset into two pieces. $\mathcal{A}(\epsilon)$ consists of all data items where *every* component of the data has a value in the range $[-(1-\epsilon), (1-\epsilon)]$. $\mathcal{B}(\epsilon)$ consists of all the rest of the data. If you think of the data set as forming a cubical orange, then $\mathcal{B}(\epsilon)$ is the rind (which has thickness ϵ) and $\mathcal{A}(\epsilon)$ is the fruit.

Your intuition will tell you that there is more rind than fruit. This is true, for three dimensional oranges, but not true in high dimensions. The fact that the orange is cubical just simplifies the calculations, but has nothing to do with the real problem.

We can compute $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$ and $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$. These probabilities tell us the probability a data item lies in the fruit (resp. rind). $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$ is easy to compute as

$$P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) = (2(1-\epsilon)))^d \left(\frac{1}{2^d}\right) = (1-\epsilon)^d$$

and

$$P(\{\mathbf{x} \in \mathcal{B}(\epsilon)\}) = 1 - P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) = 1 - (1-\epsilon)^d.$$

But notice that, as $d \to \infty$,

$$P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) \to 0.$$

This means that, for large d, we expect most of the data to be in $\mathcal{B}(\epsilon)$. Equivalently, for large d, we expect that at least one component of each data item is close to either 1 or -1.

This suggests (correctly) that much data is quite far from the origin. It is easy to compute the average of the squared distance of data from the origin. We want

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \int_{\text{box}} \left(\sum_i x_i^2\right) P(\mathbf{x}) d\mathbf{x}$$

but we can rearrange, so that

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \sum_i \mathbb{E}\left[x_i^2\right] = \sum_i \int_{\text{box}} x_i^2 P(\mathbf{x}) d\mathbf{x}.$$

Now each component of $\mathbf{x}$ is independent, so that $P(\mathbf{x}) = P(x_1)P(x_2)\ldots P(x_d)$. Now we substitute, to get

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \sum_i \mathbb{E}\left[x_i^2\right] = \sum_i \int_{-1}^{1} x_i^2 P(x_i) dx_i = \sum_i \frac{1}{2}\int_{-1}^{1} x_i^2 dx_i = \frac{d}{3},$$

so as d gets bigger, most data points will be further and further from the origin. Worse, as d gets bigger, data points tend to get further and further from one another. We can see this by computing the average of the squared distance of data points from one another. Write $\mathbf{u}$ for one data point and $\mathbf{v}$; we can compute

$$\mathbb{E}\left[d(\mathbf{u},\mathbf{v})^2\right] = \int_{\text{box}}\int_{\text{box}} \sum_i (u_i - v_i)^2 d\mathbf{u} d\mathbf{v} = \mathbb{E}\left[\mathbf{u}^T\mathbf{u}\right] + \mathbb{E}\left[\mathbf{v}^T\mathbf{v}\right] - \mathbb{E}\left[\mathbf{u}^T\mathbf{v}\right]$$

but since $\mathbf{u}$ and $\mathbf{v}$ are independent, we have $\mathbb{E}\left[\mathbf{u}^T\mathbf{v}\right] = \mathbb{E}[\mathbf{u}]^T\mathbb{E}[\mathbf{v}] = 0$. This yields

$$\mathbb{E}\left[d(\mathbf{u},\mathbf{v})^2\right] = 2\frac{d}{3}$$

meaning that, for large d, we expect our data points to be quite far apart.

It is difficult to build histogram representations for high dimensional data. The strategy of dividing the domain into boxes, then counting data into them, fails miserably because there are too many boxes. In the case of our cube, imagine we wish to divide each dimension in half (i.e. between $[-1, 0]$ and between $[0, 1]$). Then we must have 2^d boxes. This presents two problems. First, we will have difficulty representing this number of boxes. Second, unless we are exceptionally lucky, most boxes must be empty because we will not have 2^d data items.

1.5 THE MULTIVARIATE NORMAL DISTRIBUTION

All the nasty facts about high dimensional data, above, suggest that we need to use quite simple probability models. By far the most important model is the multivariate normal distribution, which is quite often known as the multivariate gaussian distribution. There are two sets of parameters in this model, the mean μ and the covariance Σ. For a d-dimensional model, the mean is a d-dimensional column vector and the covariance is a $d \times d$ dimensional matrix. The covariance is a symmetric matrix. For our definitions to be meaningful, the covariance matrix must be positive definite.

The form of the distribution $p(\mathbf{x}|\mu, \Sigma)$ is

$$p(\mathbf{x}|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right).$$

The following facts explain the names of the parameters:

Useful facts: *Parameters of a Multivariate Normal Distribution*

Assuming a multivariate normal distribution, we have

- $\mathbb{E}[\mathbf{x}] = \mu$, meaning that the mean of the distribution is μ.
- $\mathbb{E}\left[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T\right] = \Sigma$, meaning that the entries in Σ represent covariances.

Assume I know have a dataset of items $\mathbf{x}_i$, where i runs from 1 to N, and we wish to model this data with a multivariate normal distribution. The maximum likelihood estimate of the mean, $\hat{\mu}$, is

$$\hat{\mu} = \frac{\sum_i \mathbf{x}_i}{N}$$

(which is quite easy to show). The maximum likelihood estimate of the covariance, $\hat{\Sigma}$, is

$$\hat{\Sigma} = \frac{\sum_i (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T}{N}$$

(which is rather a nuisance to show, because you need to know how to differentiate a determinant). These facts mean that we already know most of what is interesting about multivariate normal distributions (or gaussians).

1.5.1 Affine Transformations and Gaussians

Gaussians behave very well under affine transformations. In fact, we've already worked out all the math. Assume I have a dataset $\mathbf{x}_i$. The mean of the maximum likelihood gaussian model is $\mathsf{mean}\left(\{\mathbf{x}_i\}\right)$, and the covariance is $\mathsf{Covmat}\left(\{\mathbf{x}_i\}\right)$. I can now transform the data with an affine transformation, to get $\mathbf{y}_i = \mathcal{A}\mathbf{x}_i + \mathbf{b}$. The mean of the maximum likelihood gaussian model for the transformed dataset is $\mathsf{mean}\left(\{\mathbf{y}_i\}\right)$, and we've dealt with this; similarly, the covariance is $\mathsf{Covmat}\left(\{\mathbf{y}_i\}\right)$, and we've dealt with this, too.

A very important point follows in an obvious way. I can apply an affine transformation to any multivariate gaussian to obtain one with (a) zero mean and (b) independent components. In turn, this means that, *in the right coordinate system*, any gaussian is a product of zero mean one-dimensional normal distributions. This fact is quite useful. For example, it means that simulating multivariate normal distributions is quite straightforward — you could simulate a standard normal distribution for each component, then apply an affine transformation.