CS168: The Modern Algorithmic Toolbox Lecture #7: Understanding Principal Component Analysis (PCA)

Tim Roughgarden & Gregory Valiant

April 20, 2015

1 Introduction

1.1 Lecture Goal

Principal components analysis (PCA) is a basic and widely used technique for exploring data. If you go on to take specialized courses in machine learning or data mining, you'll certainly hear more about it. The goal of this lecture is develop your internal mapping between the linear algebra used to describe the method and the simple geometry that explains what's really going on. Ideally, after understanding this lecture, PCA should seem almost obvious in hindsight.

1.2 A Silly Example

Let's begin with a silly example. Suppose you have a friend who is baffled by their high-dimensional data set — n points in d dimensions, where both n and d are large. Looking at the data, you notice that:

$$\begin{aligned} \mathbf{x}_1 &= (17, 4, 2, 5, \ldots,) \\ \mathbf{x}_2 &= (136, 32, 16, 40, \ldots,) \\ \mathbf{x}_3 &= (51, 12, 6, 15, \ldots,) \\ &\cdots \end{aligned}$$

This data set isn't so complicated after all — all of the points are multiples of a common vector! Rather than storing n d-dimensional points, this data set can be represented as a single d-dimensional vector and n scalars (the multiples).

If the points are only approximately all multiples of a common vector:

$$\begin{aligned}
\mathbf{x}_1 &= (17, 4, 2, 5, \ldots,) \\
\mathbf{x}_2 &= (140, 32, 14, 40, \ldots,) \\
\mathbf{x}_3 &= (52, 10, 7, 15, \ldots,) \\
&\cdots
\end{aligned}$$

it is still true that almost all of the raw data can be thrown away with little loss of information.

1.3 Goal of PCA

The goal of PCA is to approximately express each of n d-dimensional vectors[1] $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ as linear combinations of k orthogonal d-dimensional vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k \in \mathbb{R}^d$, so that

$$\mathbf{x}_i \approx \sum_{j=1}^{k} a_{ij} \mathbf{w}_j$$

for each $i = 1, 2, \ldots, n$. (In the silly example, $k = 1$.) PCA offers a formal definition of which k vectors are the "best" ones for this purpose. Next lecture, we'll see that there are also good algorithms for computing these vectors.

1.4 Motivation

There are several reasons to approximately re-express data as linear combinations of a small number of vectors.

1. *Dimensionality reduction.* PCA can be viewed as a form of lossy compression or dimensionality reduction, continuing a theme developed earlier (especially in Lecture #4). In some cases, PCA is able to throw out much of the raw data with little loss of information.

   Recall that the dimensionality reduction techniques in Lecture #4 — the Johnson-Lindenstrauss (JL) transform and MinHash — are "data-oblivious." For example, the JL transform just picks a matrix with i.i.d. Gaussian entries, without paying attention to what the actual data set is. PCA, by contrast, is all about understanding the data set at hand — the method is deterministic and the dimensions chosen are directly determined by the data. The JL transform offers approximate distance preservation guarantees for every point set — you can apply it "for free" if all you care about is interpoint Euclidean distances — but it requires a relatively large number of dimensions. PCA will work well for some data sets, even when k is tiny, and poorly for others.

[1] Representing, for example: images (dimensions = pixels); measurements (dimensions = sensors); documents (dimensions = words); and so on.

2. *De-noising.* Thus far, we've been talking about lossy compression as if it's always a bad thing. Sometimes you're actually happy to rid yourself of some of the data, if you believe it to be more noise than signal. When the data represents something simple with a modest amount of noise added, the low-dimensional approximate representation given by PCA can better illuminate the underlying phenomenon than the raw data.

3. *Data visualization.* A key point is that, for many data sets, PCA can give illuminating results even when k is 1 or 2. In these cases, the data can be plotted and inspected for interesting patterns. (If $\mathbf{x}_i$ is approximated as $\sum_{j=1}^{k} a_{ij}\mathbf{w}_j$ by PCA, then it is plotted in k-dimensional space as the point $(a_{i1}, \ldots, a_{ik})$.) We'll see a striking example of this next lecture, where raw genetic data of Europeans turn out to encode a shocking amount of geographic information. You'll also gain some experience with this on Mini-Project #4.

4. *Data interpretation.* In some cases, the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ that PCA chooses are easy to interpret. For example, if each $\mathbf{x}_i$ represents the movies watched by a Netflix customer, one might expect some of the $\mathbf{w}_i$'s to encode movie genre information.

PCA also initiates a new theme that we'll develop over several upcoming lectures: choosing the right representation for your data can expose interesting patterns that would be otherwise undetectable.

PCA has a number of "killer applications" in many different fields; one famous one in computer science is the Eigenfaces project [1]. Here, the data points are a bunch of images of faces — all framed in the same way, under the same lighting conditions. Thus d is the number of pixels (around 65K) and each dimension encodes the intensity of one pixel. It turns out that using only 100–150 vectors are enough to represent almost all of the images with high accuracy — far less than the 65K needed for exactly representing all of the images. Face recognition then boils down to a nearest-neighbor-type computation in the low-dimensional space spanned by these vectors. There have of course been lots of advances in face recognition since the 1991 publication of [1], but PCA remains a key building block in modern approaches to the problem.

2 Defining the Problem

This lecture focuses on the special case of $k = 1$, where the goal is to fit the "best" line to a data set. Solving this special case will naturally lead to all of the ideas needed to solve the case of general k, as well.

2.1 Preprocessing

Before using PCA, it's important to preprocess the data. First, the points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ should be centered around the origin, in the sense that $\sum_{i=1}^{n} \mathbf{x}_i$ is the all-zero vector. This is easy to enforce by subtracting (i.e., shifting) each point by the "sample mean" $\frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$. After

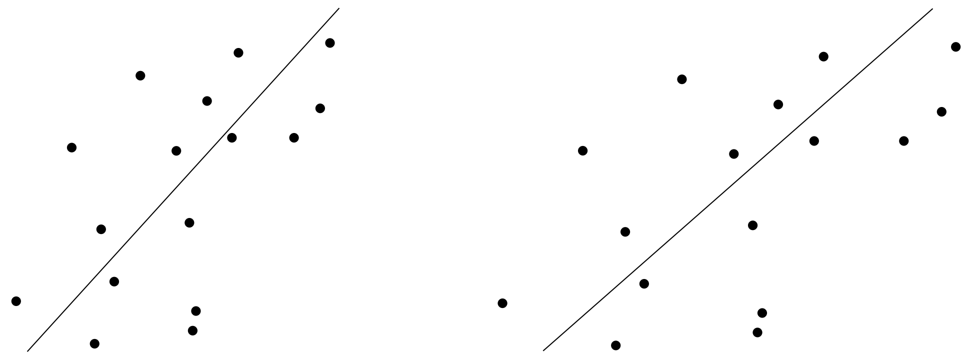

Figure 1: Scaling the x-axis yields a different best-fit line.

finding the best-fit line for the shifted point set, one simply shifts the line back by the original sample mean to get the best-fit line for the original uncentered data set. This shifting trick makes the necessary linear algebra a bit simpler and clearer.

Second, in many applications it is important to scale each coordinate appropriately (see also Mini-Project #4). The most common approach to this is: if $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is a point set centered at the origin, then for each coordinate $j = 1, 2, \ldots, d$, divide the jth coordinate of every point by the "sample deviation" in that coordinate, $\sqrt{\sum_{i=1}^n x_{ij}^2}$. The motivation for this coordinate scaling is the fact that, without it, the result of PCA would be highly sensitive to the units in which each coordinate is measured. For example, changing units from miles to kilometers in some coordinate yields the "same" data set in some sense, and yet this change would scale up all values in this coordinate, which in turn would cause a different "best-fit" line to be computed.[2] In some applications, like with images — where all coordinates are in the same units, namely pixel intensities — there is no need to do such coordinate scaling.

2.2 The Objective Function

There's more than one way to define the "best-fit line" through a point set.[3] PCA, by definition, minimizes the average squared Euclidean distance between a point and the line:

$$\operatorname*{argmin}_{\mathbf{w}\,:\,\|\mathbf{w}\|=1} \frac{1}{n} \sum_{i=1}^{n} \left((\text{distance between } \mathbf{x}_i \text{ and line spanned by } \mathbf{w})^2 \right). \tag{1}$$

Note that we're identifying a line (through the origin) with a unit vector $\mathbf{w}$ in the same direction. Minimizing the Euclidean distances between the points and the chosen line should seem natural enough; the one thing you might be wondering about is why we square these

[2]It should be geometrically clear, already in two dimensions, that stretching out one coordinate affects the best line through the points. See also Figure 1.

[3]A different definition is given by linear regression; see Mini-Project #4.

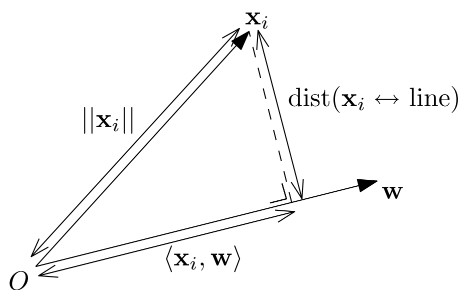
$\mathbf{x}_i$
$\text{dist}(\mathbf{x}_i \leftrightarrow \text{line})$
$||\mathbf{x}_i||$
$\mathbf{w}$
$\langle \mathbf{x}_i, \mathbf{w} \rangle$
O

Figure 2: The geometry of the inner product.

distances before adding them up. One reason is that it ensures that the best-fit line passes through the origin. Another is a tight connection to variance maximization, discussed next.

Recall the geometry of projections and the inner product (Figure 2). Recall the Pythagorean Theorem: for a right triangle with sides a and b and hypotenuse c, $a^2+b^2=c^2$. Instantiating this for the right triangle shown in Figure 2, we have

$$(\text{dist}(\mathbf{x}_i \leftrightarrow \text{line}))^2 + \langle \mathbf{x}_i, \mathbf{w} \rangle^2 = \|\mathbf{x}_i\|^2. \tag{2}$$

The right-hand side of (2) is a constant, independent of the choice of line $\mathbf{w}$. Thus, there is a zero-sum game between the squared distance between a point $\mathbf{x}_i$ and the line spanned by $\mathbf{w}$ and the squared length of the projection of $\mathbf{x}_i$ on this line — making one of these quantities bigger makes the other one smaller. This implies that the objective function of maximizing the squared projections — the *variance* of the projection of the point set — is equivalent to the original objective in (1):

$$\operatorname*{argmax}_{\mathbf{w}\,:\,\|\mathbf{w}\|=1} \frac{1}{n} \sum_{i=1}^{n} \langle \mathbf{x}_i, \mathbf{w} \rangle^2. \tag{3}$$

The objective function of maximizing variance is natural in its own right, and the fact that PCA's objective function admits multiple interpretations builds confidence that it's performing a fundamental operation on the data. For example, imagine the data points fall into two well-separated clusters; see Figure 3 for an illustration but imagine a high-dimensional version that can't be so easily eyeballed. In this case, maximizing variance corresponds to preserving the separation between the two clusters post-projection. (With a poorly chosen line, the two clusters would project on top of one another, obscuring the structure in the data.)

PCA effectively assumes that variance in the data corresponds to interesting information. One can imagine scenarios where such variance corresponds to noise rather than signal, in which case PCA may not produce illuminating results. (See next lecture for more on PCA failure cases.)

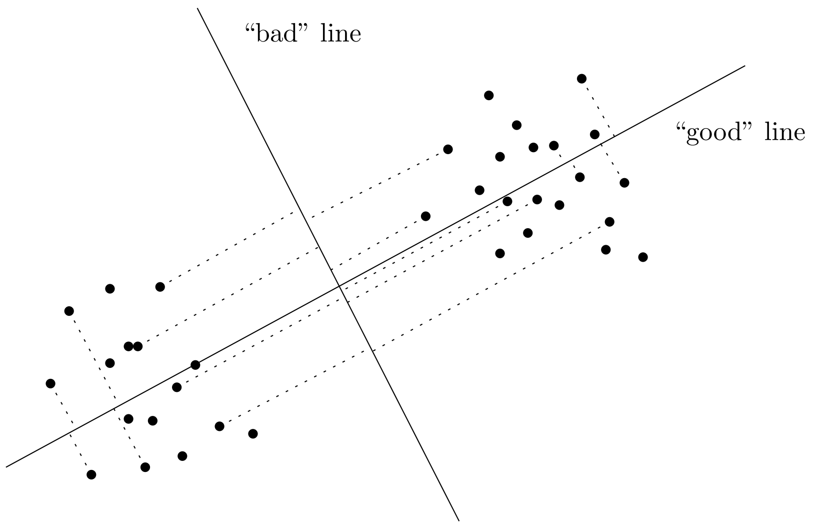

Figure 3: For the good line, the projection of the points onto the line keeps the two clusters separated, while the projection onto the bad line merges the two clusters.

3 Solving the Problem

We've now formally defined the mathematical goal of PCA (with $k = 1$): identify the direction (i.e., unit vector $\mathbf{w}$) that optimizes (1) or, equivalently, (3). How do we figure out this $\mathbf{w}$? — after all, there is an infinite number of unit vectors to try. A big reason for the popularity of PCA is that this optimization problem is easily solved using sophomore-level linear algebra. After we review the necessary preliminaries and build up your geometric intuition, the solution should seem straightforward in hindsight.

3.1 Rewriting the Optimization Problem

Let's see how to rewrite variance-maximization (3) using linear algebra. First, we take the data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ — remember these are in d-dimensional space — and write them as the rows of a $n \times d$ matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ & \vdots & \\ — & \mathbf{x}_n & — \end{bmatrix}.$$

Thus, for a unit vector $\mathbf{w} \in \mathbb{R}^d$, we have

$$\mathbf{A}\mathbf{w} = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{w} \rangle \\ \langle \mathbf{x}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{w} \rangle \end{bmatrix},$$

so $\mathbf{Aw}$ is just a column vector populated with all the projection lengths of the $\mathbf{x}_i$'s onto the line spanned by $\mathbf{w}$. We care about the sum of the squares of these (recall (3)), which motivates taking the inner product of $\mathbf{Aw}$ with itself:

$$\mathbf{w}^T\mathbf{A}^T\mathbf{A}\mathbf{w} = (\mathbf{Aw})^T(\mathbf{Aw}) = \sum_{i=1}^{n} \langle \mathbf{x}_i, \mathbf{w} \rangle^2.$$

Summarizing, our variance-maximization problem can be rephrased as

$$\operatorname*{argmax}_{\mathbf{w}\,:\,\|\mathbf{w}\|=1} \mathbf{w}^\mathbf{T}\mathbf{B}\mathbf{w}, \tag{4}$$

where $\mathbf{B}$ is a $d \times d$ matrix of the form $\mathbf{A}^\mathbf{T}\mathbf{A}$.[4] This problem is called "maximizing a quadratic form."

The matrix $\mathbf{A}^\mathbf{T}\mathbf{A}$ has a natural interpretation. The entries of the matrix are inner products of columns of $\mathbf{A}$. For example, suppose the $\mathbf{x}_i$'s represent documents, with dimensions (i.e., columns of $\mathbf{A}$) corresponding to words. Then the inner product of two columns of $\mathbf{A}$ measures how frequently the corresponding pair of words co-occur in a document. The matrix $\mathbf{A}^\mathbf{T}\mathbf{A}$ is called the *covariance* or *correlation matrix* of the $\mathbf{x}_i$'s, depending on whether or not each of the coordinates was normalized in a preprocessing step (see Section 2.1).

3.2 Solving (4): The Diagonal Case

To gain some understanding for the optimization problem (4) that PCA solves, let's begin with a very special case: where B is a diagonal matrix

$$\begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_d \end{pmatrix} \tag{5}$$

with sorted nonnegative entries $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$ on the diagonal.

There is a simple geometric way to think about diagonal matrices. A $d \times d$ matrix maps points in $\mathbb{R}^d$ back to points in $\mathbb{R}^d$ — the matrix "moves $\mathbb{R}^d$ around," in effect. For example, the matrix

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

moves each point (x, y) of the plane to the point $(2x, y)$ with double the x-coordinate and the same y-coordinate. For example, the points of the circle are mapped to the points of the "squashed circle," or ellipse, shown in Figure 4. More generally, a diagonal matrix of the form (5) can be thought of as "stretching" $\mathbb{R}^d$, with the ith axis getting stretched by the factor λ_i.

[4]We are ignoring the $\frac{1}{n}$ scaling factor in (3), because the optimal solution $\mathbf{w}$ is the same with or without it.

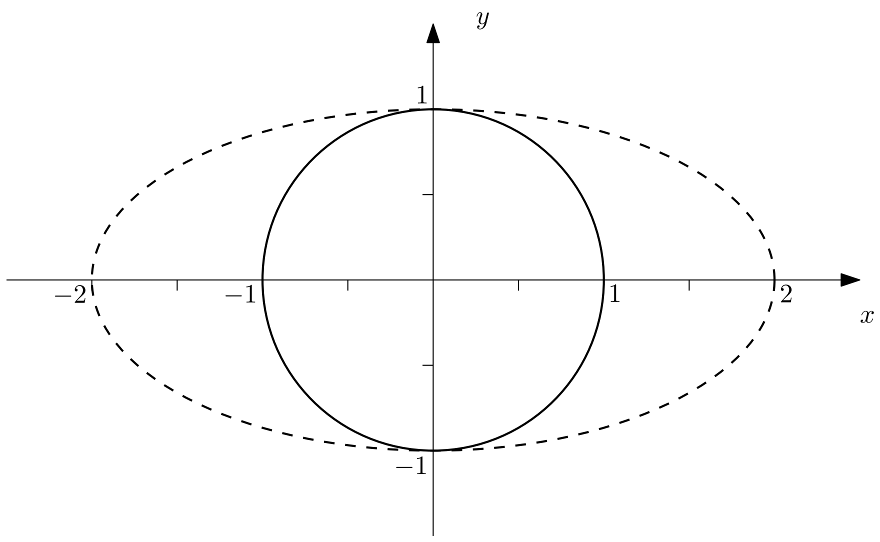

Figure 4: The point (x, y) on the unit circle is mapped to $(2x, y)$.

A natural guess for the direction $\mathbf{w}$ that maximizes $\mathbf{w^T B w}$ with $\mathbf{B}$ diagonal is the "direction of maximum stretch," namely $\mathbf{w} = \mathbf{e}_1$, where $\mathbf{e}_1 = (1, 0, \ldots, 0)$ denotes the first standard basis vector. (Recall $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_d \geq 0$.) To verify the guess, let $\mathbf{w}$ be an arbitrary unit vector, and write

$$\mathbf{w}^T(\mathbf{B}\mathbf{w}) = \begin{pmatrix} w_1 & w_2 & \cdots & w_d \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 w_1 \\ \lambda_2 w_2 \\ \vdots \\ \lambda_d w_d \end{pmatrix} = \sum_{i=1}^{d} w_i^2 \lambda_i. \tag{6}$$

Since $\mathbf{w}$ is a unit vector, the w_i^2's sum to 1. Thus $\mathbf{w^T B w}$ is always an average of the λ_i's, with the averaging weights given by the w_i^2's. Since λ_1 is the biggest λ_i, the way to make this average as large as possible is to set $w_1 = 1$ and $w_i = 0$ for $i > 1$. That is, $\mathbf{w} = \mathbf{e}_1$ maximizes $\mathbf{w^T B w}$, as per our guess.

3.3 Diagonals in Disguise

Let's generalize our solution in Section 3.2 by considering matrices $\mathbf{B}$ that, while not diagonal, are really just "diagonals in disguise." Geometrically, what we mean is that $\mathbf{B}$ still does nothing other than stretch out different orthogonal axes, possibly with these axes being a "rotated version" of the original ones. See Figure 5 for a rotated version of the previous example, which corresponds to the matrix

$$\begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{\text{rotate back } 45^\circ} \cdot \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}}_{\text{stretch}} \cdot \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{\text{rotate clockwise } 45^\circ} \tag{7}$$

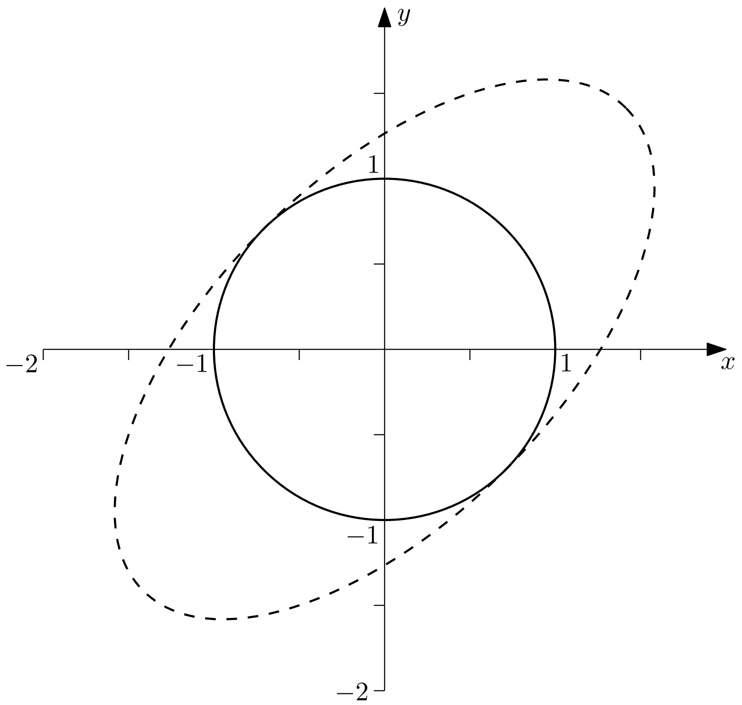

Figure 5: The same scaling as Figure 4, but now rotated 45 degrees.

So what's a "rotated diagonal" in higher dimensions? The appropriate generalization of a rotation is an *orthogonal matrix*.[5] Recall that an orthogonal matrix $\mathbf{Q}$ is a square matrix where the columns are a set of orthonormal vectors — that is, each column has unit length, and the inner product of two different columns is 0. A key property of orthogonal matrices is that they preserve length — that is, $\|\mathbf{Qv}\| = \|\mathbf{v}\|$ for every vector $\mathbf{v}$. We review briefly why this is: since the columns of $\mathbf{Q}$ are orthonormal vectors, we have

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I},$$

which means that the inverse of an orthogonal matrix is simply its transpose. (For example, see the two inverse rotation matrices in (7).) Then, we have

$$\|\mathbf{Qv}\|^2 = (\mathbf{Qv})^T(\mathbf{Qv}) = \mathbf{v}^T\mathbf{Q}^T\mathbf{Q}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2,$$

showing that $\mathbf{Qv}$ and $\mathbf{v}$ have same norm.

Now consider a matrix $\mathbf{B}$ that can be written $\mathbf{B} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$ for an orthogonal matrix $\mathbf{Q}$ and diagonal matrix $\mathbf{D}$ as in (5) — this is what we mean by a "diagonal in disguise." Such a matrix $\mathbf{B}$ has a "direction of maximum stretch" — the (rotated) axis that gets stretched the most (i.e., by λ_1). Since the direction of maximum stretch under $\mathbf{D}$ is $\mathbf{e}_1$, the direction of maximum stretch under $\mathbf{B}$ is the direction that gets mapped to $\mathbf{e}_1$ under $\mathbf{Q}$ — which is

[5] In addition to rotations, orthogonal matrices capture operations like reflections and permutations of the coordinates.

$\mathbf{Q}^{-1}\mathbf{e}_1$ or, equivalently, $\mathbf{Q}^T\mathbf{e}_1$. Notice that $\mathbf{Q}^T\mathbf{e}_1$ is simply the first column of $\mathbf{Q}^T$ — the first row of $\mathbf{Q}$.

This direction of maximum stretch is again the solution to the variance-maximization problem (3). To see this, first plug in this choice $\mathbf{w}_1 = \mathbf{Q}^T\mathbf{e}_1$ to obtain

$$\mathbf{w}_1^T\mathbf{B}\mathbf{w}_1 = \mathbf{w}_1^T\mathbf{Q}^T\mathbf{D}\mathbf{Q}\mathbf{w}_1 = \mathbf{e}_1^T\mathbf{D}\mathbf{e}_1 = \lambda_1.$$

Second, for every unit vector $\mathbf{w}$, $\mathbf{Q}\mathbf{w}$ is also a unit vector (since $\mathbf{Q}$ is orthogonal), so $\mathbf{w}^T\mathbf{Q}^T\mathbf{D}\mathbf{Q}\mathbf{w}$ is an average of the λ_i's, just as in (6) (with averaging weights given by the squared coordinates of $\mathbf{Q}\mathbf{w}$, rather than those of $\mathbf{w}$). Thus $\mathbf{w}^{\mathbf{T}}\mathbf{B}\mathbf{w} \leq \lambda_1$ for every $\mathbf{w}$, implying that $\mathbf{w}_1 = \mathbf{Q}^T\mathbf{e}_1$ maximizes $\mathbf{w}^{\mathbf{T}}\mathbf{B}\mathbf{w}$.

3.4 The General Case

We've seen that when the matrix $\mathbf{B}$ can be written as $\mathbf{Q}^T\mathbf{D}\mathbf{Q}$ for an orthogonal matrix $\mathbf{Q}$ and diagonal matrix $\mathbf{D}$, it's easy to understand how to maximize the variance (3): the optimal solution is to set $\mathbf{w}$ equal to the first row of $\mathbf{Q}$, and geometrically this is just the direction of maximum stretch when we view $\mathbf{B}$ as a map from $\mathbb{R}^d$ to itself. But we don't want to solve the problem (3) only for diagonals in disguise — we want to solve it for an arbitrary covariance matrix $\mathbf{B} = \mathbf{A}^T\mathbf{A}$. Happily, we've already done this: recall from linear algebra that *every matrix* $\mathbf{B}$ *of the form* $\mathbf{A}^T\mathbf{A}$ *can be written* $\mathbf{B} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$ *for an orthogonal matrix* $\mathbf{Q}$ *and diagonal matrix* $\mathbf{D}$ *as in* (5).[6]

3.5 Eigenvectors and Eigenvalues

The bottom line of the derivation above is that *PCA boils down to computing the eigenvectors of the covariance matrix* $\mathbf{A}^{\mathbf{T}}\mathbf{A}$. Recall that an *eigenvector* of a matrix $\mathbf{B}$ is a vector $\mathbf{v}$ that is stretched in the same direction by $\mathbf{B}$, meaning $\mathbf{B}\mathbf{v} = \lambda\mathbf{v}$ for some $\lambda \neq 0$. The value λ is the corresponding *eigenvalue*. Eigenvectors are just the "axes of stretch" of the geometric discussions above.

When we write $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ as $\mathbf{B} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$, we're actually writing the matrix in terms of its eigenvectors and eigenvalues. The ith row of $\mathbf{Q}$ is an eigenvector of $\mathbf{B}$ with eigenvalue λ_i. (Proof: the ith row of $\mathbf{Q}$ can be written $\mathbf{Q}^T\mathbf{e}_i$, and since $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$ we have $\mathbf{B}\mathbf{Q}^T\mathbf{e}_i = \mathbf{Q}^T\mathbf{D}\mathbf{e}_i = \lambda_i\mathbf{Q}^T\mathbf{e}_i$.) We already determined that the first row of $\mathbf{Q}$ is PCA's answer — we now see that this is simply the *first principal eigenvector* of the matrix $\mathbf{A}^T\mathbf{A}$, meaning the eigenvector with the largest eigenvalue. We'll discuss how to efficiently compute eigenvectors in the next lecture.

[6]If you look back at your notes from your linear algebra class, the most likely relevant statement is that every symmetric matrix can be diagonalized by orthogonal matrices. (The proof is not obvious, but it is covered in standard linear algebra courses.) Here we're also using that $\mathbf{B}$ is "positive semidefinite" — because it is not only symmetric but also has the form $\mathbf{A}^T\mathbf{A}$, it follows that $\mathbf{w}^{\mathbf{T}}\mathbf{B}\mathbf{w} \geq 0$ for every $\mathbf{w}$, which in turn implies that all of the diagonal entries of $\mathbf{D}$ must be nonnegative (as you should check).

3.6 Larger Values of k

The discussion so far focused on the $k = 1$ case (fitting a line to the data), but the case of larger k is almost the same. For general k, the objective functions of minimizing the squares of distances to a k-dimensional subspace and of maximizing the variance of the projections onto a k-dimensional subspace are again equivalent. The solution to the variance-maximization problem is now to pick the k orthogonal axes that are stretched the most. The first direction is picked as in the $k = 1$ case; the second direction is, among all those orthogonal to the first, that of maximum stretch; and so on. These are just the first k rows of the matrix $\mathbf{Q}$ used to decompose $\mathbf{B}$, so the solution is just the top k eigenvectors of the covariance matrix $\mathbf{A}^{\mathbf{T}}\mathbf{A}$.

3.7 Interpreting the Results

At the very beginning of lecture, we said that the goal is to approximately express each data point $\mathbf{x}_i$ as a linear combination of k vectors (in the same space, $\mathbb{R}^d$). PCA chooses the top k eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$ for this purpose. The corresponding approximate linear combination is then

$$\mathbf{x}_i \approx \sum_{j=1}^{k} \langle \mathbf{x}_i, \mathbf{w}_j \rangle \mathbf{w}_j.$$

Both the $\mathbf{w}_j$'s and the projections of the $\mathbf{x}_i$'s onto them can be interesting. Here are some things to look at:

1. Look at the data points with the largest (most positive) and smallest (most negative) projections $\langle \mathbf{x}_i, \mathbf{w}_1 \rangle$ on the first principal component. Does this suggest a potential "meaning" for the component? Are there data points clustered together at either of the two ends, and if so, what do these have in common?

2. Associate each data point $\mathbf{x}_i$ with two coordinate values, corresponding to the top two principal components ($\langle \mathbf{x}_i, \mathbf{w}_1 \rangle$ and $\langle \mathbf{x}_i, \mathbf{w}_2 \rangle$). Plot all points according to their two coordinate values. Do any interesting clusters pop out, for example in any of the four corners? By looking at pairs that have similar second coordinates — both pairs with similar first coordinates, and pairs with very different first coordinates — it is often possible to obtain a rough interpretation of the second principal component.

3. Looking at the coordinates of $\mathbf{w}_1$ — the linear combination of the original data point attributes that it uses — can also be helpful. For example, in the Eigenfaces application mentioned earlier, one principal component, when viewed as image in its own right, basically corresponds to a mustache!

References

[1] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.