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

Tim Roughgarden & Gregory Valiant*

April 18, 2016

## 1 A Toy Example

The following toy example gives a sense of the problem solved by principal component analysis (PCA) and many of the reasons why you might want to apply it to a data set — to visualize the data in a lower-dimensional space, to understand the sources of variability in the data, and to understand correlations between different coordinates of the data points.

Suppose you ask some friends to rate, on a scale of 1 to 10, four different foods: kale salad, Taco Bell, sashimi, and pop tarts, with the results shown in Table 1. So in our usual

| | kale | taco bell | sashimi | pop tarts |
|---|---|---|---|---|
| Alice | 10 | 1 | 2 | 7 |
| Bob | 7 | 2 | 1 | 10 |
| Carolyn | 2 | 9 | 7 | 3 |
| Dave | 3 | 6 | 10 | 2 |

Table 1: Your friends' ratings of four different foods.

notation, $m$ (the number of data points) is 4, and $n$ (the number of dimensions) is also 4. Note that, in contrast to the linear regression problem discussed last week, the data points do not have "labels" of any sort. Can we usefully visualize this data set in fewer than 4 dimensions? Can we understand the forces at work behind the differences in opinion of the various foods?

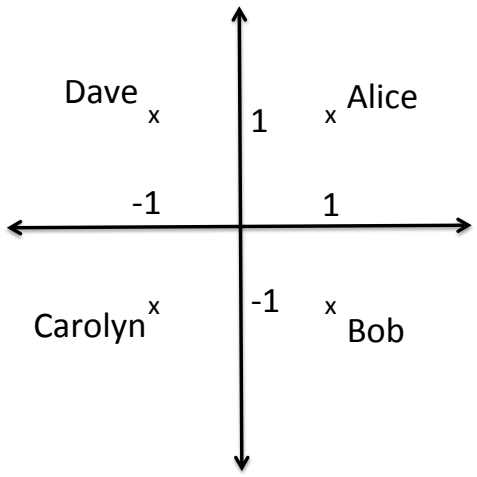



Figure 1: Visualizing 4-dimensional data in the plane.

The key observation is that each row (data point) can be approximately expressed as

$$\bar{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2, \tag{1}$$

where

$$\bar{\mathbf{x}} = (5.5, 4.5, 5, 5.5)$$

is the average of the data points,

$$\mathbf{v}_1 = (3, -3, -3, 3),$$

$$\mathbf{v}_2 = (1, -1, 1, -1),$$

and $(a_1, a_2)$ is (1,1) for Alice, (1,-1) for Bob, (-1,-1) for Carolyn, and (-1,1) for Dave (Figure 1). For example, $\mathbf{x} + \mathbf{v}_1 + \mathbf{v}_2 = (9.5, 0.5, 3, 7.5)$, which is approximately equal to Alice's scores. Similarly, $\mathbf{x} - \mathbf{v}_1 - \mathbf{v}_2 = (1.5, 8.5, 7, 3.5)$, which is approximately equal to Carolyn's scores.

What use is such an approximation? One answer is that it provides a way to visualize the data points, as in Figure 1. With more data points, we can imagine inspecting the resulting figure for clusters of similar data points.

A second answer is that it helps interpret the data. We think of each data point as having a "$v_1$-coordinate" (i.e., $a_1$) and a "$v_2$-coordinate" ($a_2$). What does the vector $v_1$ "mean?" The first and fourth coordinates both have a large positive value (and so are positively correlated) while the second and third coordinates have a large negative value (and so are positively correlated with each other and negatively correlated with the first and fourth coordinates). Noting that kale salad and pop tarts are vegetarian[1] while Taco Bell and sashimi are not, we can interpret the $v_1$ coordinate as indicating the extent to which someone has vegetarian preferences.[2] By similar reasoning, we can interpret the second vector $v_2$ as indicating the

[1]Little-known fact: pop tarts — the unfrosted kinds, at least — are inadvertently vegan.

[2]Conversely, if we knew which friends were vegetarian and which were not, we could deduce which foods are most likely to be vegetarian.

extent to which someone is health-conscious. The fact that $v_1$ has larger coordinates than $v_2$ indicates that it is the stronger of the two effects.

We'll see that the vectors $v_1$ and $v_2$, once normalized, correspond to the "top two principal components" of the data. The point of PCA is to compute approximations of the type (1) automatically, including for large data sets.

## 1.1 Goal of PCA

The goal of PCA is to approximately express each of $m$ $n$-dimensional vectors[3] $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ as linear combinations of $k$ $n$-dimensional vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$, so that

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

for each $i = 1, 2, \ldots, m$. (In the toy example, $m = n = 4$ and $k = 2$.) 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.

## 1.2 Relation to Dimensionality Reduction

The high-level goal of PCA should remind you of a couple of topics studied in previous lectures. First, recall Johnson-Lindenstrauss (JL) dimensionality reduction (Lecture #4), where the goal is also to re-express a bunch of high-dimensional points as low-dimensional points. Recall that this method maps a point $\mathbf{x}$ to a point $(\langle \mathbf{x}, \mathbf{r}_1 \rangle, \ldots, \langle \mathbf{x}, \mathbf{r}_k \rangle)$, where each $\mathbf{r}_j$ is a random unit vector (independent standard Gaussians in each coordinate, then normalized).[4] The same $\mathbf{r}_j$'s are used for all data points. PCA and JL dimensionality reduction are different in the following respects.

1. JL dimensionality reduction assumes that you care about the Euclidean distance between each pair of points — it is designed to approximately preserve these distances. PCA offers no guarantees about preserving pairwise distances.

2. JL dimensionality reduction is data-oblivious, in the sense that the random vectors $\mathbf{r}_1, \ldots, \mathbf{r}_k$ are chosen without looking at the data points. PCA, by contrast, is deterministic, and the whole point is to compute vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ that "explain the data."

3. For the above reason, the $k$ coordinates used in JL dimensionality reduction have no intrinsic meaning, while those used in PCA are often meaningful (recall the toy example).

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

[4] The notation $\langle \mathbf{x}, \mathbf{y} \rangle$ indicates the inner (or dot) product $\sum_{i=1}^{n} x_i y_i$ of the two vectors.

4. JL dimensionality reduction generally only gives good results when $k$ is at least in the low hundreds. On the other hand, this value of $k$ works for every data set of a given size. PCA can give meaningful results even when $k$ is 1 or 2, but there are also data sets where it provides little interesting information (for any $k$).

### 1.3 Relation to Linear Regression

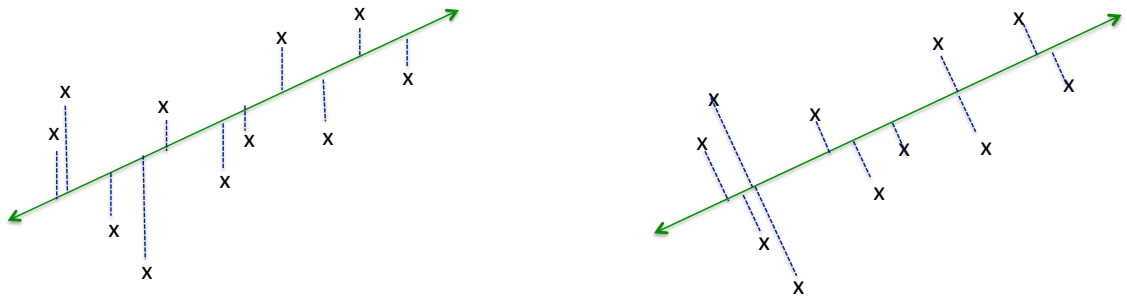
Both PCA and linear regression (see Lectures #5 and #6) concern fitting the "best" $k$-dimensional subspace to a point set. One difference between the two methods is in the interpretation of the input data. With linear regression, each data point has a real-valued label, which can be interpreted as a special "label coordinate." The goal of linear regression is to learn the relationship between this special coordinate and the other coordinates. This makes sense when the "label" is a dependent variable that is approximately a linear combination of all of the other coordinates (the independent variables). In PCA, by contrast, all coordinates are treated equally, and they are not assumed to be independent from one another. This makes sense when there is a set of latent (i.e., hidden/underlying) variables, and all of the coordinates of your data are (approximately) linear combinations of those variables.

Second, PCA and linear regression use different definitions of "best fit." Recall that in linear regression the goal is to minimize the total squared error, where the error on a data point is the difference between the linear function's prediction and the actual label of the data point. With one-dimensional data (together with labels), this corresponds to computing the line that minimizes the sum of the squared vertical distances between the line and the data points (Figure 2(a)). This reflects the assumption that the coordinate corresponding to labels is the important one. We define the PCA objective function formally below; for two-dimensional data, the goal is to compute the line that minimizes the sum of the squared *perpendicular* distances between the line and the data points (Figure 2(b)). This reflects the assumption that all coordinates play a symmetric role and so the usual Euclidean distance is most appropriate (e.g., rotating the point set just results in the "best-fit" line being rotated accordingly). On Mini-Project #4 you will learn more about which situations call for PCA and which call for linear regression.

## 2 Defining the Problem

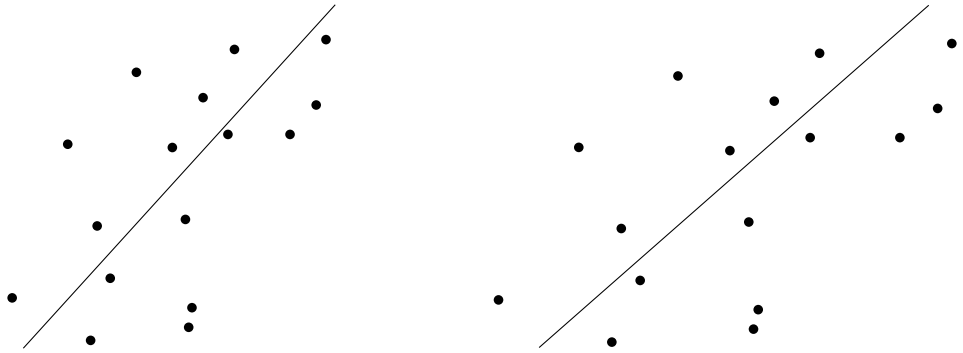
### 2.1 Preprocessing

Before using PCA, it's important to preprocess the data. First, the points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ should be centered around the origin, in the sense that $\sum_{i=1}^m \mathbf{x}_i$ is the all-zero vector. This is easy to enforce by subtracting (i.e., shifting) each point by the "sample mean" $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^m \mathbf{x}_i$. (In the toy example, we had $\bar{\mathbf{x}} = (5.5, 4.5, 5, 5.5)$.) After 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) Linear regression (b) PCA

Figure 2: Linear regression minimizes the sum of squared vertical distances, while PCA minimizes the sum of squared perpendicular distances.



(a) (b)

Figure 3: Scaling the $x$-axis yields a different best-fit line.

simpler and clearer.

Second, in many applications it is important to scale each coordinate appropriately. The most common approach to this is: if $\mathbf{x}_1, \ldots, \mathbf{x}_m$ is a point set centered at the origin, then for each coordinate $j = 1, 2, \ldots, n$, divide the $j$th coordinate of every point by the "sample deviation" in that coordinate, $\sqrt{\sum_{i=1}^{m} x_{ij}^2}$. 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.[5] 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. (The same goes for the toy example in Section 1.)

[5]It should be geometrically clear, already in two dimensions, that stretching out one coordinate affects the best line through the points. See also Figure 3.

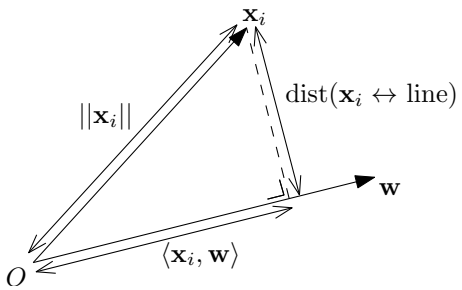



Figure 4: The geometry of the inner product.

## 2.2 The Objective Function

For clarity, we first discuss the special case of $k = 1$, where the goal is to fit the "best" line to a data set. Everything in this section generalizes easily to the case of general $k$ (Section 2.3).

As foreshadowed above, PCA defines the "best-fit line" as the one that minimizes the average squared Euclidean distance between the line and the data points:

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

Note that we're identifying a line (through the origin) with a unit vector $\mathbf{v}$ 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 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 4). In particular, $\langle \mathbf{x}_i, \mathbf{v} \rangle$ *is the (signed) length of the projection of* $\mathbf{x}_i$ *onto the line spanned by* $\mathbf{v}$. 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 4, we have

$$(\text{dist}(\mathbf{x}_i \leftrightarrow \text{line}))^2 + \langle \mathbf{x}_i, \mathbf{v} \rangle^2 = \|\mathbf{x}_i\|^2. \tag{3}$$

The right-hand side of (3) is a constant, independent of the choice of line $\mathbf{v}$. Thus, there is a zero-sum game between the squared distance between a point $\mathbf{x}_i$ and the line spanned by $\mathbf{v}$ 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 (2):

$$\operatorname*{argmax}_{\mathbf{v}\,:\,\|\mathbf{v}\|=1} \frac{1}{m} \sum_{i=1}^{m} \langle \mathbf{x}_i, \mathbf{v} \rangle^2. \tag{4}$$

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

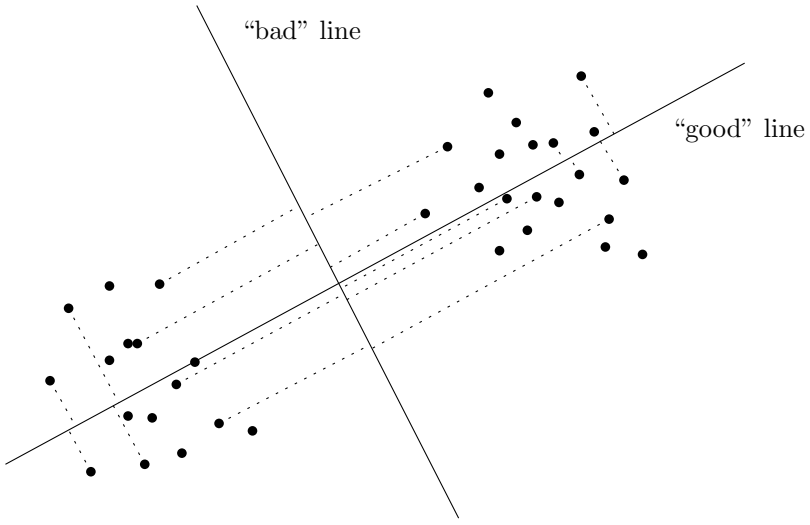



Figure 5: For the good line, the projection of the points onto the line keeps the two clusters separated, while the projection onto the bad line merges the two clusters.

fall into two well-separated clusters; see Figure 5 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 Section 4 for more on PCA failure cases.)

## 2.3 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. So the PCA objective is now

$$\underset{k\text{-dimensional subspaces } S}{\operatorname{argmax}} \frac{1}{m}\sum_{i=1}^{m}(\text{length of } \mathbf{x}_i\text{'s projection on } S)^2. \tag{5}$$

To rephrase this objective in a more convenient form, recall the following basic linear algebra. First, vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are *orthonormal* if they all have unit length ($\|\mathbf{v}_i\|_2 = 1$ for all $i$) and are orthogonal ($\langle \mathbf{v}_i, \mathbf{v}_j \rangle = 0$ for all $i \neq j$). For example, the standard basis vectors (of the form $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$) are orthonormal. Rotating these vectors gives more set of orthonormal vectors.

The *span* of a collection $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$ of vectors is all of their linear combinations: $\{\sum_{j=1}^k \lambda_j \mathbf{v}_j : \lambda_1, \ldots, \lambda_k \in \mathbb{R}\}$. If $k = 1$, then this span is a line through the origin; if $k = 2$ and $\mathbf{v}_1, \mathbf{v}_2$ are linearly independent (i.e., not multiples of each other) then the span is a plane (through the origin); and so on.

One nice fact about orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is that the squared projection length of a point onto the subspace spanned by the vectors is just the sum of the squares of its projections onto each of the vectors:

$$(\text{length of } \mathbf{x}_i\text{'s projection on span}(\mathbf{v}_1, \ldots, \mathbf{v}_k))^2 = \sum_{j=1}^k \langle \mathbf{x}_i, \mathbf{v}_j \rangle^2. \tag{6}$$

(For intuition, consider first the case where $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are all standard basis vectors, so $\langle \mathbf{x}_i, \mathbf{v}_j \rangle$ just picks out one coordinate of $\mathbf{x}_i$). This identity is *not* true if the $\mathbf{v}_j$'s are not orthonormal (e.g., imagine if $k = 2$ and $\mathbf{v}_1, \mathbf{v}_2$ are linearly independent but nearly the same vector).

Combining (5) and (6), we can state the objective of PCA for general $k$ in its standard form: compute orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ to maximize

$$\frac{1}{m} \sum_{i=1}^m \underbrace{\sum_{j=1}^k \langle \mathbf{x}_i, \mathbf{v}_j \rangle^2}_{\text{squared projection length}}. \tag{7}$$

The resulting $k$ orthonormal vectors are called the *top $k$ principal components* of the data.[6]

To recap, the formal definition of the problem solved by PCA is:

> *given* $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \mathbb{R}^n$ *and a parameter* $k \geq 1$*, compute orthonormal vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$ *to maximize* (7)*.*

Next lecture we discuss computational methods for finding the top $k$ principal components. We conclude this lecture with use cases and "non-use cases" (i.e., failure modes) of PCA. Given a black-box that computes principal components, what would you do with it?

# 3 Use Cases

## 3.1 Data Visualization

### 3.1.1 A General Recipe

PCA is commonly used to visualize data. In this use case, one typically takes $k$ to be 1, 2, or 3. Given data points $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$, here is the approach:

[6] In the toy example, the two vectors $(3, -3, -3, 3)$ and $(1, -1, 1, -1)$, once normalized to be unit vectors, are close to the top two principal components. The actual principal components have messier numbers so we don't report them here. Try computing them with one line of Matlab code!

1. Perform PCA to get the top $k$ principal components $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$.

2. For each data point $\mathbf{x}_i$, define its "$\mathbf{v}_1$-coordinate" as $\langle \mathbf{x}_i, \mathbf{v}_1 \rangle$, its "$\mathbf{v}_2$-coordinate" as $\langle \mathbf{x}_i, \mathbf{v}_2 \rangle$, and so on. This associates $k$ coordinates with each data point $\mathbf{x}_i$, according to the extent to which it points in the same direction as each of the top $k$ principal components. (So a large positive coordinate means that $\mathbf{x}_i$ is close to the same direction as $\mathbf{v}_j$, while a large negative coordinate means that $\mathbf{x}_i$ points in the opposite direction.)

3. Plot the point $\mathbf{x}_i$ in $\mathbb{R}^k$ as the point $(\langle \mathbf{x}_i, \mathbf{v}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{v}_k \rangle)$.[7]

This is more or less what we did with our toy example in Figure 1, although the scaling was different because our vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ were not unit vectors.

What is the "meaning" of the point $(\langle \mathbf{x}_i, \mathbf{v}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{v}_k \rangle)$? Recall that one way to think about PCA is as a method for approximating a bunch of data points as linear combinations of the same $k$ vectors. PCA uses the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ for this purpose, and the coordinates $(\langle \mathbf{x}_i, \mathbf{v}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{v}_k \rangle)$ specify the linear combination of these vectors that most closely approximate $\mathbf{x}_i$ (with respect to Euclidean distance). That is, PCA approximates each $\mathbf{x}_i$ as

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

### 3.1.2 Interpreting the Results

Both the $\mathbf{v}_j$'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 $\langle \mathbf{x}_i, \mathbf{v}_1 \rangle$ 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. Plot all points according to their $k$ coordinate values (i.e., projection lengths onto the top $k$ principal components). Do any interesting clusters pop out (e.g., with $k = 2$, in any of the four corners)? For example, 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 a principal component — the linear combination of the original data point attributes that it uses — can also be helpful. (E.g., see the Eigenfaces application below.)

[7] Alternatively, it can sometimes be illuminating to plot points using a subset of the top $k$ principal components — the first and third components, say.

### 3.1.3 Do Genomes Encode Geography?

Can you infer where someone grew up from their DNA? In a remarkable study, Novembre et al. [1] used PCA to show that in some cases the answer is "yes." This is a case study in how PCA can reveal that the data "knows" things that you wouldn't expect.

Novembre et al. [1] considered a data set comprising 1387 Europeans (the rows). Each person's genome was examined in the same 200,000 "SNPs" (the columns) — positions that tend to exhibit gene mutation.[8] So in one of these positions, maybe 90% of all people have a "C," while 10% have an "A." In a nutshell, Novembre et al. [1] apply PCA to this data set (with $m \approx 1400$, $n \approx 200,000$, and $k = 2$) and plot the data according to the top two principal components $\mathbf{v}_1$ and $\mathbf{v}_2$ (using the exact same recipe as in Section 3.1.1, with the $x$- and $y$-axes corresponding to $\mathbf{v}_1$ and $\mathbf{v}_2$).[9]

We emphasize that this plot depends on genomic data *only*. To interpret it, the authors then color-coded each plotted point according to the country of origin of the corresponding person (this information was available in the meta-data). Check out the resulting plot in [1], available from the course Web page — it's essentially a map of Europe! The first principal component $\mathbf{v}_1$ corresponds roughly to the latitude of where someone's from, and $\mathbf{v}_2$ to the longitude (rotated 16° counter-clockwise). That is, *their genome "knows" where they're from!* This suggests that many Europeans have bred locally for long enough for geographic information to be reflected in their DNA.

Is this conclusion obvious in hindsight? Not really. For example, if you repeat the experiment using U.S. data, the top principal components correspond more to waves of immigration than to anything geographic. So in this case the top principal components reflect temporal rather than spatial effects.

There are competing mathematical models for genetic variation. Some of these involve spatial decay, while others are based on multiple discrete and well-differentiated groups. The study of Novembre et al. [1] suggests that the former models are much more appropriate for Europeans.

## 3.2 Data Compression

One famous "killer application" of PCA in computer science is the Eigenfaces project [2]. Here, the data points are a bunch of images of faces — all framed in the same way, under the same lighting conditions. Thus $n$ is the number of pixels (around 65K) and each dimension encodes the intensity of one pixel. It turns out that using only the top 100–150 principal components is enough to represent almost all of the images with high accuracy — far less than the 65K needed for exactly representing all of the images.[10] Each of these principal components can be viewed as an image and in many cases interpreted directly. (By contrast, in the previous example, we couldn't interpret the top two principal components by inspec-

[8] You'll work with a similar data set on Mini-Project #4.

[9] As usual with PCA, for best results non-trivial preprocessing of the data was necessary.

[10] We'll see next lecture that principal components are in fact eigenvectors of a suitable matrix — hence the term "Eigenfaces."

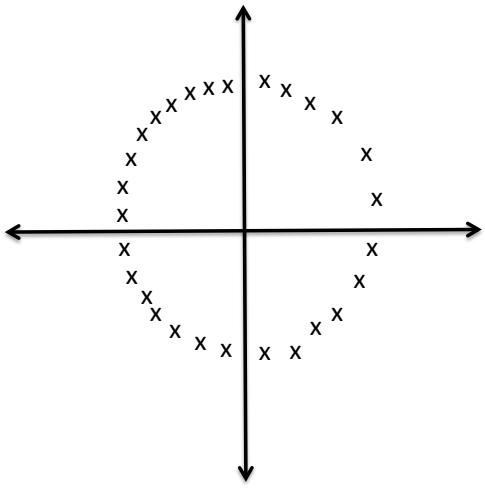


Figure 6: GPS data from a ferris wheel. PCA is not designed to discover nonlinear structure.

tion — we needed to plot the data points first.) For example, one principal component may indicate the presence or absence of glasses, another the presence or absence of a mustache, and so on. (See the figures in [2], available from the course Web site.) These techniques have been used in face recognition, which boils down to a nearest-neighbor-type computation in the low-dimensional space spanned by the top principal components. There have of course been lots of advances in face recognition since the 1991 publication of [2], but PCA remains a key building block in modern approaches to the problem.

# 4 Failure Cases

When and why does PCA fail?

1. You messed up the scaling/normalizing. PCA is sensitive to different scalings/normalizations of the coordinates. Much of the artistry of getting good results from PCA involves choosing an appropriate scaling for the different data coordinates, and perhaps additional preprocessing of the data (like outlier removal).

2. Non-linear structure. PCA is all about finding linear structure in your data. If your data has some low-dimensional, but non-linear structure, e.g., you have two data coordinates, $x, y$, with $y \approx x^2$, then PCA will not find this.[11] Similarly for GPS data from someone on a ferris wheel (Figure 6) — the data is one dimensional (each point can be specified by an angle) but this dimension will not be identified by PCA.

3. Non-orthogonal structure. It is nice to have coordinates that are interpretable, like in our toy example (Section 1) and our Europe example (Section 3.1.3). Even if your data

[11] As mentioned in the context of linear regression last lecture, you can always add extra non-linear dimensions to your data, like coordinates $x^2, y^2, xy$, etc., and then PCA this higher-dimensional data. This idea doesn't always work well with PCA, however.

is essentially linear, the fact that the principal components are all orthogonal will often mean that after the top few components, it will be almost impossible to interpret the meanings of the components. In the Europe example, the third and later components are forced to orthogonal to the latitude and longitude components. This is a rather artificial constraint to put on an interpretable feature.

# References


[1] John Novembre, Toby Johnson, Katarzyna Bryc, Zoltn Kutalik, Adam R. Boyko, Adam Auton, Amit Indap, Karen S. King, Sven Bergmann, Matthew R. Nelson, Matthew Stephens, and Carlos D. Bustamante. Genes mirror geography within Europe. *Nature*, 456:98–101, 2008.

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

# CS168: The Modern Algorithmic Toolbox
# Lecture #8: How PCA Works

Tim Roughgarden & Gregory Valiant*

April 20, 2016

## 1 Introduction

Last lecture introduced the idea of principal components analysis (PCA). The definition of the method is, for a given data set and parameter $k$, to compute the $k$-dimensional subspace (through the origin) that minimizes the average squared distance between the points and the subspace, or equivalently that maximizes the variance of the projections of the data points onto the subspace. We talked about a couple of common use cases. The first is data visualization, where one uses the top few principal components as a new coordinate axis and plot the projections of all of the data points in this new coordinate system. We saw a remarkable example using European genomic data, where such a plot showed that geographic information is embedded in the participants' DNA. The second use case discussed was data compression, with the "Eigenfaces" project being a canonical example, where taking $k$ in the range of 100 or 150 is large enough to closely approximate a bunch of images (65,000-dimensional data). Finally, we mentioned some of the primary weaknesses of PCA: it can get tricked by high-variance noise, it fails to discover nonlinear structure, and the orthogonality constraints on the principal components mean that principal components after the first few can be difficult to interpret.

Today's lecture is about how PCA actually works — that is, how to actually compute the top $k$ principal components of a data set. Along the way, we'll 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.

# 2 Characterizing Principal Components

## 2.1 The Setup

Recall that the input to the PCA method is $m$ $n$-dimensional data points $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ and a parameter $k \in \{1, 2, \ldots, n\}$. We assume that the data is centered, meaning that $\sum_{i=1}^m \mathbf{x}_i$ is the all-zero vector. (This can be enforced by subtracting out the sample mean $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^m \mathbf{x}_i$ in a preprocessing step.) The output of the method is defined as $k$ orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ — the "top $k$ principal components" — that maximize the objective function

$$\frac{1}{m}\sum_{i=1}^{m} \underbrace{\sum_{j=1}^{k} \langle \mathbf{x}_i, \mathbf{v}_j \rangle^2}_{\text{squared projection length}}. \tag{1}$$

But how would one actually solve this optimization problem and compute the $\mathbf{v}_j$'s? Even with $k = 1$, 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.

## 2.2 Rewriting the Optimization Problem

To develop the solution, we first consider only the $k = 1$ case. We'll see that the case of general $k$ reduces to this case (Section 2.6). With $k = 1$, the objective function is

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

Let's see how to rewrite variance-maximization (2) using linear algebra. First, we take the data points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ — remember these are in $n$-dimensional space — and write them as the rows of an $m \times n$ matrix $\mathbf{X}$:

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

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

$$\mathbf{X}\mathbf{v} = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{v} \rangle \\ \langle \mathbf{x}_2, \mathbf{v} \rangle \\ \vdots \\ \langle \mathbf{x}_m, \mathbf{v} \rangle \end{bmatrix},$$

so $\mathbf{Xv}$ is just a column vector populated with all the projection lengths of the $\mathbf{x}_i$'s onto the line spanned by $\mathbf{v}$. We care about the sum of the squares of these (recall (2)), which motivates taking the inner product of $\mathbf{Xv}$ with itself:

$$\mathbf{v}^\top \mathbf{X}^\top \mathbf{X}\mathbf{v} = (\mathbf{Xv})^\top (\mathbf{Xv}) = \sum_{i=1}^{m} \langle \mathbf{x}_i, \mathbf{v} \rangle^2.$$

Summarizing, our variance-maximization problem can be rephrased as

$$\operatorname*{argmax}_{\mathbf{v}\,:\,\|\mathbf{v}\|=1} \mathbf{v}^\top \mathbf{A}\mathbf{v}, \tag{3}$$

where $\mathbf{A}$ is an $n \times n$ matrix of the form $\mathbf{X}^\top\mathbf{X}$.[1] This problem is called "maximizing a quadratic form."

The matrix $\mathbf{X}^\top\mathbf{X}$ has a natural interpretation.[2] The $(i, j)$ entry of this matrix is the inner product of the $i$th row of $\mathbf{X}^\top$ and the $j$th column of $\mathbf{X}$ — i.e., of the $i$th and $j$th columns of $\mathbf{X}$. So $\mathbf{X}^\top\mathbf{X}$ just collects the inner products of columns of $\mathbf{X}$, and is a symmetric matrix. For example, suppose the $\mathbf{x}_i$'s represent documents, with dimensions (i.e., columns of $\mathbf{X}$) corresponding to words. Then the inner product of two columns of $\mathbf{X}$ measures how frequently the corresponding pair of words co-occur in a document.[3] The matrix $\mathbf{X}^\top\mathbf{X}$ 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 (as discussed in Lecture #7).

## 2.3 Solving (3): The Diagonal Case

To gain some understanding for the optimization problem (3) that PCA solves, let's begin with a very special case: where $\mathbf{A}$ is a diagonal matrix

$$\begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ & & \ddots \\ 0 & & & \lambda_n \end{pmatrix} \tag{4}$$

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

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

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

[1] We are ignoring the $\frac{1}{m}$ scaling factor in (2), because the optimal solution $\mathbf{v}$ is the same with or without it.

[2] Recall that this matrix also appeared in the normal equations, the closed-form solution to the ordinary least squares problem (Lecture #6).

[3] So after centering such an $\mathbf{X}$, frequently co-occurring pairs of words correspond to positive entries of $\mathbf{X}^\top\mathbf{X}$ and pairs of words that almost never appear together are negative entries.

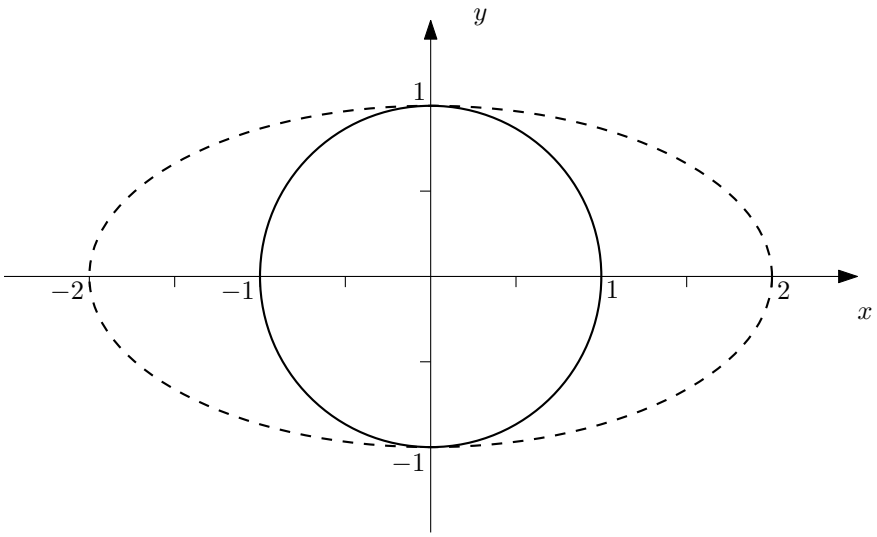



Figure 1: The point $(x, y)$ on the unit circle is mapped to $(2x, y)$.

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 $\{(x, y) : x^2+y^2 = 1\}$ are mapped to the points $\{\frac{x^2}{2^2} + y^2 = 1\}$ of the ellipse shown in Figure 1. More generally, a diagonal matrix of the form (4) can be thought of as "stretching" $\mathbb{R}^n$, with the $i$th axis getting stretched by the factor $\lambda_i$, and the unit circle being mapped to the corresponding "ellipsoid" (i.e., the analog of an ellipse in more than 2 dimensions).

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

$$\mathbf{v}^\top(\mathbf{A}\mathbf{v}) = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 v_1 \\ \lambda_2 v_2 \\ \vdots \\ \lambda_n v_n \end{pmatrix} = \sum_{i=1}^{n} v_i^2 \lambda_i. \tag{5}$$

Since $\mathbf{v}$ is a unit vector, the $v_i^2$'s sum to 1. Thus $\mathbf{v}^\top \mathbf{A}\mathbf{v}$ is always an average of the $\lambda_i$'s, with the averaging weights given by the $v_i^2$'s. Since $\lambda_1$ is the biggest $\lambda_i$, the way to make this average as large as possible is to set $v_1 = 1$ and $v_i = 0$ for $i > 1$. That is, $\mathbf{v} = \mathbf{e}_1$ maximizes $\mathbf{v}^\top \mathbf{A}\mathbf{v}$, as per our guess.

## 2.4 Diagonals in Disguise

Let's generalize our solution in Section 2.3 by considering matrices $\mathbf{A}$ that, while not diagonal, are really just "diagonals in disguise." Geometrically, what we mean is that $\mathbf{A}$ still does nothing other than stretch out different orthogonal axes, possibly with these axes being a "rotated version" of the original ones. See Figure 2 for a rotated version of the previous

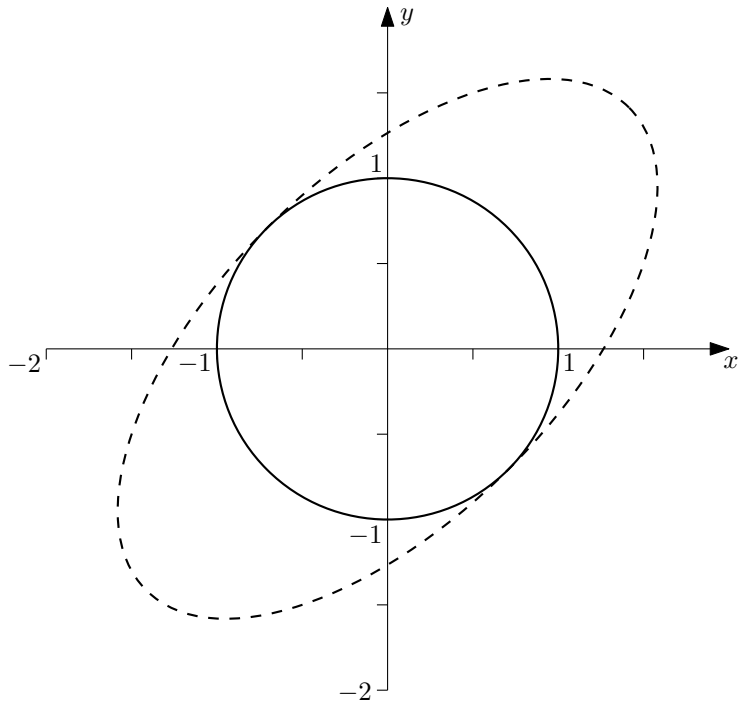



Figure 2: The same scaling as Figure 1, but now rotated 45 degrees.

example, which corresponds to the matrix

$$\begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{\text{rotate back } 45^\circ} \cdot \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}}_{\text{stretch}} \cdot \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{\text{rotate clockwise } 45^\circ} . \tag{6}$$

So what's a "rotated diagonal" in higher dimensions? The appropriate generalization of a rotation is an *orthogonal matrix*.[4] 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, $\|\mathbf{Q}\mathbf{v}\| = \|\mathbf{v}\|$ for every vector $\mathbf{v}$. We review briefly why this is: since the columns of $\mathbf{Q}$ are orthonormal vectors, we have

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

since the $(i, j)$ entry of $\mathbf{Q}^\top \mathbf{Q}$ is just the inner product of the $i$th and $j$th columns of $\mathbf{Q}$. (Taking transposes, we also get $\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$. This shows that $\mathbf{Q}^\top$ is also an orthogonal matrix, or equivalently that the rows of an orthogonal matrix are orthonormal vectors.) This means that the inverse of an orthogonal matrix is simply its transpose. (For example, see the two inverse rotation matrices in (6).) Then, we have

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

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

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

Now consider a matrix $\mathbf{A}$ that can be written $\mathbf{A} = \mathbf{QDQ}^\top$ for an orthogonal matrix $\mathbf{Q}$ and diagonal matrix $\mathbf{D}$ as in (4) — this is what we mean by a "diagonal in disguise." Such a matrix $\mathbf{A}$ has a "direction of maximum stretch" — the (rotated) axis that gets stretched the most (i.e., by $\lambda_1$). Since the direction of maximum stretch under $\mathbf{D}$ is $\mathbf{e}_1$, the direction of maximum stretch under $\mathbf{A}$ is the direction that gets mapped to $\mathbf{e}_1$ under $\mathbf{Q}^\top$ — which is $(\mathbf{Q}^{-1})^\top\mathbf{e}_1$ or, equivalently, $\mathbf{Qe}_1$. Notice that $\mathbf{Qe}_1$ is simply the first column of $\mathbf{Q}$ — the first row of $\mathbf{Q}^\top$.

This direction of maximum stretch is again the solution to the variance-maximization problem (2). To see this, first plug in this choice $\mathbf{v}_1 = \mathbf{Qe}_1$ to obtain

$$\mathbf{v}_1^\top\mathbf{A}\mathbf{v}_1 = \mathbf{v}_1^\top\mathbf{QDQ}^\top\mathbf{v}_1 = \mathbf{e}_1^\top\mathbf{Q}^\top\mathbf{QDQ}^\top\mathbf{Qe}_1 = \mathbf{e}_1^\top\mathbf{De}_1 = \lambda_1.$$

Second, for every unit vector $\mathbf{v}$, $\mathbf{Q}^\top\mathbf{v}$ is also a unit vector (since $\mathbf{Q}$ is orthogonal), so $\mathbf{v}^\top\mathbf{QDQ}^\top\mathbf{v}$ is an average of the $\lambda_i$'s, just as in (5) (with averaging weights given by the squared coordinates of $\mathbf{Q}^\top\mathbf{v}$, rather than those of $\mathbf{v}$). Thus $\mathbf{v}^\top\mathbf{Av} \leq \lambda_1$ for every unit vector $\mathbf{v}$, implying that $\mathbf{v}_1 = \mathbf{Qe}_1$ maximizes $\mathbf{v}^\top\mathbf{Av}$.

## 2.5 General Covariance Matrices

We've seen that when the matrix $\mathbf{A}$ can be written as $\mathbf{QDQ}^\top$ for an orthogonal matrix $\mathbf{Q}$ and diagonal matrix $\mathbf{D}$, it's easy to understand how to maximize the variance (2): the optimal solution is to set $\mathbf{v}$ equal to the first row of $\mathbf{Q}^\top$, and geometrically this is just the direction of maximum stretch when we view $\mathbf{A}$ as a map from $\mathbb{R}^n$ to itself. But we don't want to solve the problem (2) only for diagonals in disguise — we want to solve it for an arbitrary covariance matrix $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$. Happily, we've already done this: recall from linear algebra that *every matrix* $\mathbf{A}$ *of the form* $\mathbf{X}^\top\mathbf{X}$ *can be written as* $\mathbf{A} = \mathbf{QDQ}^\top$ *for an orthogonal matrix* $\mathbf{Q}$ *and diagonal matrix* $\mathbf{D}$ *as in* (4).[5]

Summarizing, we've shown the following:

> when $k = 1$, the solution to (2) is the first row of $\mathbf{Q}^\top$, where $\mathbf{X}^\top\mathbf{X}^\top = \mathbf{QDQ}^\top$ with $\mathbf{Q}$ orthonormal and $\mathbf{D}$ diagonal with sorted diagonal entries.

## 2.6 Larger Values of $k$

The solution to the variance-maximization problem (1) is analogous, namely to pick the $k$ orthogonal axes that are stretched the most by $\mathbf{A}$. Extending the derivation above gives:

[5] 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.) For some symmetric matrices $\mathbf{A}$, the corresponding diagonal matrix $\mathbf{D}$ will have some negative entries. For symmetric matrices of the form $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$, however, all of the diagonal entries are nonnegative. Such matrices are called "positive semindefinite." To see this fact, note that (i) if $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$, then $\mathbf{v}^\top\mathbf{Av} = (\mathbf{Xv})^\top\mathbf{Xv} \geq 0$ for every $\mathbf{v}$; and (ii) if $\mathbf{A} = \mathbf{QDQ}^T$ with the $i$th diagonal entry of $\mathbf{D}$ negative, then taking $\mathbf{v} = \mathbf{Qe}_i$ provides a vector with $\mathbf{v}^\top\mathbf{Av} < 0$ (why?).

for general $k$, the solution to (1) is the first $k$ rows of $\mathbf{Q}^\top$, where $\mathbf{X}^\top\mathbf{X} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$ with $\mathbf{Q}$ orthonormal and $\mathbf{D}$ diagonal with sorted diagonal entries.

Note that since $\mathbf{Q}$ is orthonormal, the first $k$ rows of $\mathbf{Q}^\top$ are indeed a set of $k$ orthonormal vectors, as required in (1).[6] For a matrix $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$, these are called the *top $k$ principal components of* $\mathbf{X}$.

## 2.7 Eigenvectors and Eigenvalues

Now that we've characterized the top $k$ principal components as the first $k$ rows of $\mathbf{Q}^\top$ in the decomposition $\mathbf{X}^\top\mathbf{X} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, how can we actually compute them? The answer follows from a reinterpretation of these vectors as eigenvectors of the covariance matrix $\mathbf{X}^\top\mathbf{X}$.

Recall that an *eigenvector* of a matrix $\mathbf{A}$ is a vector $\mathbf{v}$ that is stretched in the same direction by $\mathbf{A}$, meaning $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ for some $\lambda \in \mathbb{R}$. The value $\lambda$ is the corresponding *eigenvalue*. Eigenvectors are just the "axes of stretch" of the geometric discussions above, with eigenvalues giving the stretch factors.[7]

When we write $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$ as $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, we're actually writing the matrix in terms of its eigenvectors and eigenvalues — the rows of $\mathbf{Q}^\top$ are the eigenvectors of $\mathbf{A}$, and the diagonal entries in $\mathbf{D}$ are the corresponding eigenvalues. To see this, consider the $i$th row of $\mathbf{Q}^\top$, which can be written $\mathbf{Q}\mathbf{e}_i$. Since $\mathbf{Q}^\top\mathbf{Q} = \mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$ we have, $\mathbf{A}\mathbf{Q}\mathbf{e}_i = \mathbf{Q}\mathbf{D}\mathbf{e}_i = \lambda_i\mathbf{Q}\mathbf{e}_i$. Hence the $i$th row of $\mathbf{Q}^\top$ is the eigenvector of $\mathbf{A}$ with eigenvalue $\lambda_i$. Thus:

> *PCA boils down to computing the $k$ eigenvectors of the covariance matrix $\mathbf{X}^\top\mathbf{X}$ that have the largest eigenvalues.*

You will often see the sentence above given as the definition of the PCA method; after this lecture there should be no mystery as to where the definition comes from.

Are the top $k$ principal components of $\mathbf{X}$ — the $k$ eigenvectors of $\mathbf{X}^\top\mathbf{X}$ that have the largest eigenvalues — uniquely defined? More or less, but there is some fine print. First, the set of diagonal entries in the matrix $\mathbf{D}$ — the set of eigenvalues of $\mathbf{X}^\top\mathbf{X}$ — is uniquely defined. Recall that we're ordering the coordinates so that these entries are in sorted order. If these eigenvalues are all distinct, then the matrix $\mathbf{Q}$ is also unique (up to a sign flip in each column).[8] If an eigenvalue occurs with multiplicity $d$, then there is a $d$-dimensional subspace of corresponding eigenvectors. Any orthonormal basis of this subspace can be chosen as the corresponding principal components.

---

[6] An alternative way to arrive at the same $k$ vectors is: choose $\mathbf{v}_1$ as the variance-maximizing direction (as in (2)); choose $\mathbf{v}_2$ as the variance-maximizing direction orthogonal to $\mathbf{v}_1$; choose $\mathbf{v}_3$ as the variance-maximizing direction orthogonal to both $\mathbf{v}_1$ and $\mathbf{v}_2$; and so on.

[7] Eigenvectors and eigenvalues will reappear in Lectures #11–12 on spectral graph theory, where they are unreasonably effective at illuminating network structure. They also play an important role in the analysis of Markov chains (Lecture #14).

[8] If $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$ and $\hat{\mathbf{Q}}$ is $\mathbf{Q}$ with all signs flipped in some column, then $\mathbf{A} = \hat{\mathbf{Q}}\mathbf{D}\hat{\mathbf{Q}}^\top$ as well.

# 3 The Power Iteration Method

## 3.1 Methods for Computing Principal Components

How does one compute the top $k$ principal components? One method uses the "singular value decomposition (SVD);" we'll talk about this in detail next lecture. (The SVD can be computed in roughly cubic time, which is fine for medium-size but not for large-scale problems.) We conclude this lecture with a description of a second method, *power iteration*. This method is popular in practice, especially in settings where one only wants the top few eigenvectors (as is often the case in PCA applications). In Matlab there are optimized and ready-to-use implementations of both the SVD and power iteration methods.

## 3.2 The Algorithm

We first describe how to use the power iteration method to compute the first eigenvector (the one with largest eigenvalue), and then explain how to use it to find the rest of them. To understand the geometric intuition behind the method, recall that if one views the matrix $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$ as a function that maps the unit sphere to an ellipsoid, then the longest axis of the ellipsoid corresponds to the top eigenvector of $\mathbf{A}$ (Figures 1 and 2). Given that the top eigenvector corresponds to the direction in which multiplication by $\mathbf{A}$ stretches the vector the most, it is natural to hope that if we start with a random vector $\mathbf{v}$, and keep applying $\mathbf{A}$ over and over, then we will end up having stretched $\mathbf{v}$ so much in the direction of $\mathbf{A}$'s top eigenvector that the image of $\mathbf{v}$ will lie almost entirely in this same direction. For example, in Figure 2, applying $\mathbf{A}$ twice (rotate/stretch/rotate-back/rotate/stretch/rotate-back) will stretch the ellipsoid twice as far along the southwest-northeast direction (i.e., along the first eigenvector). Further applications of $\mathbf{A}$ will make this axis of stretch even more pronounced. Eventually, almost all of the points on the original unit circle get mapped to points that are very close to this axis.

Here is the formal statement of the power iteration method:

**Algorithm 1**
POWER ITERATION

Given matrix $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$:

- Select random unit vector $\mathbf{u}_0$
- For $i = 1, 2, \ldots$, set $\mathbf{u}_i = \mathbf{A}^i \mathbf{u}_0$. If $\mathbf{u}_i / ||\mathbf{u}_i|| \approx \mathbf{u}_{i-1} / ||\mathbf{u}_{i-1}||$, then return $\mathbf{u}_i / ||\mathbf{u}_i||$.

Often, rather than computing $\mathbf{A}\mathbf{u}_0, \mathbf{A}^2\mathbf{u}_0, \mathbf{A}^3\mathbf{u}_0, \mathbf{A}^4\mathbf{u}_0, \mathbf{A}^5\mathbf{u}_0, \ldots$ one instead uses repeated squaring and computes $\mathbf{A}\mathbf{u}_0, \mathbf{A}^2\mathbf{u}_0, \mathbf{A}^4\mathbf{u}_0, \mathbf{A}^8\mathbf{u}_0, \ldots$. (This replaces a larger number of matrix-vector multiplications with a smaller number of matrix-matrix multiplications.)

## 3.3 The Analysis

To show that the power iteration algorithm works, we first establish that if $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, then $\mathbf{A}^i = \mathbf{Q}\mathbf{D}^i\mathbf{Q}^\top$ — that is, $\mathbf{A}^i$ has the same orientation (i.e. eigenvectors) as $\mathbf{A}$, but all of the eigenvalues are raised to the $i$th power (and are hence exaggerated—e.g. if $\lambda_1 > 2\lambda_2$, then $\lambda_1^{10} > 1000\lambda_2^{10}$).

**Claim 3.1** *If* $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, *then* $\mathbf{A}^i = \mathbf{Q}\mathbf{D}^i\mathbf{Q}^\top$.

*Proof:* We prove this via induction on $i$. The base case, $i = 1$ is immediate. Assuming that the statement holds for some specific $i \geq 1$, consider the following:

$$\mathbf{A}^{i+1} = \mathbf{A}^i \cdot \mathbf{A} = \mathbf{Q}\mathbf{D}^i \underbrace{\mathbf{Q}^\top\mathbf{Q}}_{=\mathbf{I}} \mathbf{D}\mathbf{Q}^\top = \mathbf{Q}\mathbf{D}^i\mathbf{D}\mathbf{Q}^\top = \mathbf{Q}\mathbf{D}^{i+1}\mathbf{Q}^\top,$$

where we used our induction hypothesis to get the second equality, and the third equality follows from the orthogonality of $\mathbf{Q}$. ■

We can now quantify the performance of the power iteration algorithm:

**Theorem 3.2** *With probability at least* $1/2$ *over the choice of* $\mathbf{u}_0$, *for and* $t \geq 1$,

$$|\langle \mathbf{A}^t\mathbf{u}_0, \mathbf{v}_1\rangle| \geq 1 - 2\sqrt{n}\left(\frac{\lambda_2}{\lambda_1}\right)^t,$$

*where* $\mathbf{v}_1$ *is the top eigenvector of* $\mathbf{A}$, *with eigenvalue* $\lambda_1$, *and* $\lambda_2$ *is the second-largest eigenvalue of* $\mathbf{A}$.

This result shows that the number of iterations required scales as $\frac{\log n}{\log(\lambda_1/\lambda_2)}$. The ratio $\lambda_1/\lambda_2$ is an important parameter called the *spectral gap* of the matrix $\mathbf{A}$. The bigger the spectral gap, the more pronounced is the direction of maximum stretch (compared to other axes of stretch). If the spectral gap is large, then we are in excellent shape. If $\lambda_1 \approx \lambda_2$, then the algorithm might take a long time (or might never) find $\mathbf{v}_1$.[9]

For example, if $t > 10\frac{\log n}{\log(\lambda_1/\lambda_2)}$ then $|\langle \mathbf{u}_t, \mathbf{v}_1\rangle| > 1 - 2e^{-20} > 0.99999$, and so $\mathbf{u}_t$ is essentially pointing in the same direction as $\mathbf{v}_1$.

The "with probability 1/2" statement in Theorem 3.2 can be strengthened to "with probability at least $1 - 1/2^{100}$" by repeating the above algorithm 100 times (for independent choices of $\mathbf{u}_0$), and outputting the recovered vector $\mathbf{u}$ that maximizes $\|\mathbf{A}\mathbf{u}\|$.

*Proof of Theorem 3.2:* Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ denote the eigenvectors of $\mathbf{A}$, with associated eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots$. These vectors form an orthonormal basis for $\mathbb{R}^n$. Write the random initial vector $\mathbf{u}_0 = \sum_{j=1}^n c_j\mathbf{v}_j$ in terms of this basis. We claim that, with probability

[9] If $\lambda_1 = \lambda_2$, then $\mathbf{v}_1$ and $\mathbf{v}_2$ are not uniquely defined — there is a two-dimensional subspace of eigenvectors with this eigenvalue. In this case, the power iteration algorithm will simply return a vector that lies in this subspace, which is the correct thing to do.

at least $1/2$, $|c_1| > 1/2\sqrt{n}$. This follows straightforwardly from a computation, using the fact that we can choose a random unit vector by selecting each coordinate independently from a Gaussian of variance $1/n$, and then normalizing (by a factor that will be very close to 1).

Given that $|c_1| > 1/2\sqrt{n}$,

$$|\langle \mathbf{A}^t \mathbf{u}_0, \mathbf{v}_1 \rangle| = \frac{c_1 \lambda_1^t}{\sqrt{\sum_{i=1}^n (\lambda_i^t c_i)^2}} \geq \frac{c_1 \lambda_1^t}{\sqrt{c_1^2 \lambda_1^{2t} + n\lambda_2^{2t}}} \geq \frac{c_1 \lambda_1^t}{c_1 \lambda_1^t + \lambda_2^t \sqrt{n}} \geq 1 - 2\sqrt{n} \left(\frac{\lambda_2}{\lambda_1}\right)^t.$$

■

## 3.4 Computing Further Principal Components

Applying the power iteration algorithm to the covariance matrix $\mathbf{X}^\top \mathbf{X}$ of a data matrix $\mathbf{X}$ finds (a close approximation to) the top principal component of $\mathbf{X}$. We can reuse the same method to compute subsequent principal components one-by-one, up to the desired number $k$. Specifically, to find the top $k$ principal components:

1. Find the top component, $\mathbf{v}_1$, using power iteration.

2. Project the data matrix orthogonally to $\mathbf{v}_1$:

$$\begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ & \vdots & \\ — & \mathbf{x}_m & — \end{bmatrix} \mapsto \begin{bmatrix} — & (\mathbf{x}_1 - \langle \mathbf{x}_1, \mathbf{v}_1 \rangle \mathbf{v}_1) & — \\ — & (\mathbf{x}_2 - \langle \mathbf{x}_2, \mathbf{v}_1 \rangle \mathbf{v}_1) & — \\ & \vdots & \\ — & (\mathbf{x}_m - \langle \mathbf{x}_m, \mathbf{v}_1 \rangle \mathbf{v}_1) & — \end{bmatrix}.$$

   This corresponds to subtracting out the variance of the data that is already explained by the first principal component $\mathbf{v}_1$.

3. Recurse by finding the top $k-1$ principal components of the new data matrix.

The correctness of this greedy algorithm follows from the fact that the $k$-dimensional subspace that maximizes the norms of the projections of a data matrix $X$ contains the $(k-1)$-dimensional subspace that maximizes the norms of the projections.

## 3.5 How to Choose $k$?

How do you know how many principal components are "enough"? For data visualization, often you just want the first few. In other applications, like compression, the simple answer is that you don't. In general, it is worth computing a lot of them and plotting their eigenvalues. Often the eigenvalues become small after a certain point — e.g., your data might have 200 dimensions, but after the first 50 eigenvalues, the rest are all tiny. Looking at this plot might give you some heuristic sense of how to choose the number of components so as to maximize the signal of your data, while preserving the low-dimensionality.